Klassifiseringsmetoder i høyere ordens IDS.

Diplomoppgave av
John Bildøy
Stian Clausen
Tor-Erik Klausen

Hovedoppgave til mastergraden
i informasjons- og kommunikasjonsteknologi

Høgskolen i Agder
Fakultet for teknologi
Grimstad, 30. mai, 2004

Sammendrag

Vi ser stadig en mer aggressiv Internet-verden, med en jevn økning i datakriminalitet i form av hacker-angrep og virus. Dette skaper behov for overvåkning av nettverk i form av innbruddsdeteksjonsystem (IDS).

Når angrep oppdages på nettverket vil innbruddsdeteksjonsystemet generere en alarm til nettverksansvarlig. Signaturbaserte IDS, som er den vanligste formen for IDS i dag, har et generelt problem at de genererer endel alarmer uten at det forekommer angrep. Dette kalles falske alarmer.

Vi presenterer en løsning på dette problemet med å innføre et «høyere ordens IDS». Med dette mener vi en automatisk klassifisering av alarmer fra et signaturbasert IDS på om disse er falske eller reelle. Til dette har vi benyttet maskinlæringsmetoden naiv Bayes klassifikator.

Et viktig mål har vært å minske antall falske alarmer, samtidig som man vil unngå å klassifisere reelle alarmer som falske alarmer. I vårt testmiljø hos Telenor Sikkerhetssenter har vi jobbet med alarmer generert av Snort, et typisk signaturbasert IDS. Vi oppnår gode resultater i vår testing og reduserer antall alarmer med opptil 90%.

De viktigste konklusjonene fra oppgaven vår er at et «høyere ordens IDS» vil være en god løsning til å minske antall falske alarmer.

Vi anser våre hypoteser, *«alarmer fra et signaturbasert IDS har de egenskaper som skal til for å kunne klassifiseres ved hjelp av maskinlæring»*, *«falske alarmer må kunne kjennes igjen i stor grad ved bruk av maskinlæring»* og *«feilklassifiserte reelle alarmer vil kunne reduseres kraftig ved ulike metoder innenfor maskinlæring»* som styrket utav vårt arbeid.

Forord

Denne hovedoppgaverapporten avslutter 5-årig masterstudium ved Høgskolen i Agder innen IKT. Oppgaven «Klassifiseringsmetoder i høyere ordens IDS» er gitt av Telenor sikkerhetssenter og er hovedsakelig utført i Telenors lokaler i Nygaten 4 i Arendal.

Vi vil benytte anledningen å takke våre veiledere fra høgskolen Ole-Christoffer Granmo og Nils Ulltveit-Moe. Vi vil også takke ansatte ved Telenor sikkerhetssenter for innspill og samarbeidsvilje, og da i særdeleshet Christian Flørenes.

John Bildøy, Stian Clausen & Tor-Erik Klausen.
Grimstad 30. mai, 2004.

Innholdsliste

Liste av tabeller

Kapittel 1

Innledning

1.1 Bakgrunn for oppgaven

Et viktig tiltak for øke datasikkerhet er å overvåke nettet mot misbruk. Et av flere verktøy for dette er innbruddsdeteksjonsystemer (IDS). Innbruddsdeteksjonsystemets funksjon er å monitorere nettverkstrafikk for å gjenkjenne eventuelle angrep. Signaturbasert innbruddsdeteksjon benytter seg av faste regler (signaturer) for gjenkjenning av angrepsmønstre. Angrep som oppdages av innbruddsdeteksjonsystemet vil sende alarm til nettverksansvarlig eller annet analysepersonell, som har ansvaret for å gjøre nødvendig tiltak.

Signaturbasert IDS benytter seg av faste regler (signaturer) for gjenkjenning av angrepsmønstre og viser seg å ha en god evne til å oppdage angrep, men vil generelt generere mange falske alarmer. Med falske alarmer mener vi tilfeller der innbruddsdeteksjonsystemet sender alarm uten at det er noe faktisk angrep mot nettverket. I motsatt tilfelle, der en alarm indikerer et virkelig angrep kaller vi dette for en reell alarm.

De viktigste årsakene til falske alarmer i signaturbaserte IDS:

- Signaturene er ofte generelle. Dette fører til at signaturene i enkelte tilfeller også vil reagere på normal trafikk. Grunnen til generelle signaturer kan være en sikring mot å overse angrep, da dette er farligere enn å risikere ekstra falske alarmer. Generelle signaturer kan også være en strategi for at flere beslektede angrep kan oppdages ved samme signatur .

- Signaturene kan være forenklinger av kjente mønstre for om mulig også detektere nyere versjoner av et angrep.

- Nye signaturene kan være unøyaktig som følge av hastverk, eller forfattet av ukyndige «eksperter». Om et nytt angrep opptrer kan signaturer gjerne taes i bruk før denne er grundig testet. En kan også

nevne at en enkelt kan få tak i tredjeparts-signaturer med ubekreftet kvalitet gjennom Internet-verdenen.

Problemet med falske alarmer er at de kan oversvømme antall reelle alarmer. Dette fører til et stort arbeidspress på analysepersonell som har ansvaret for å overvåke og behandle IDS-alarmer. Behandlingen kan bestå av å gjøre nødvendig tiltak for å beskytte nettverket når angrep oppstår, og vil variere veldig utfra nettverk og hvilket type angrep det er snakk om. Når presset øker på analysepersonell vil det i tillegg kunne resultere i feil og unøyaktig behandling av de alarmer som representerer faktiske angrep. Når vi i tillegg vet at et vellykket hacker-angrep kan medføre voldsomme skader på kort tid, ville det vært en fordel å kunne detektere de reelle angrepene så fort som mulig.

Det er derfor ønskelig å kunne redusere falske alarmer som genereres fra signaturbaserte innbruddsdeteksjonsystemer. En løsning på dette er å kombinere et signaturbasert innbruddsdeteksjonssystem med en automatisk klassifiseringsmetode som gjenkjenner og sorterer ut falske alarmer. Dette har vi i vår oppgave kalt for et «høyere ordens innbruddsdeteksjonsystem».

Målet med et «høyere ordens IDS» er altså å redusere falske alarmer og dermed lette arbeidet ved manuell behandling av alarmer.

Av klassifiseringsmetoder har vi i vår oppgave valgt å fokusere på maskinlæringsteknologi. Dette går i korthet ut på å automatisk lære en modul til å klassifisere en viss type av instanser, ved å presentere en rekke ferdig klassifiserte eksempler av samme type instanser. Dette blir mye brukt innen forskjellige former for kunstig intelligens.

I vårt «høyere ordens IDS» lar vi det signaturbaserte IDS monitorere nettverket og fange opp alarmer. Slike systemer har en god evne til å oppdage angrep, noe som er en fordel vi vil gjerne ta med oss. Vi utvider dette ved et maskinlært filter som ligger mellom mellom IDS og manuell analyse med den hensikt å oppdage og filtrere bort falske alarmer.

1.2 Hypoteser

Som motivasjon for oppgaven har vi følgende hypoteser vi ønsker å teste:

Hypotese 1. *Maskinlæring kan benyttes til å klassifisere alarmer fra signaturbaserte innbruddsdeteksjonssystem som falske eller reelle alarmer.*

Hypotese 2. *Maskinlæring kan i stor grad forhindre falske alarmer i et regelbasert innbruddsdeteksjonssystem.*

Hypotese 3. *Reelle alarmer som oversees som følge av feilklassifisering vil kunne reduseres kraftig ved metoder innenfor maskinlæring. Systemet vil likevel kunne gi en akseptabel reduksjon av falske alarmer.*

Første hypotese sier vi skal undersøke om alarmer fra et signaturbasert IDS har de egenskaper som skal til for å kunne klassifiseres ved hjelp av maskinlæring. Dette innebærer at elementene i alarmene er nyttbare til å bruke i maskinlæring. Hypotesen sier også at maskinlæring skal kunne klassifisere signaturbasere IDS alarmer hvorvidt disse er falsk alarm eller reell alarm.

Hypotese 2 sier at falske alarmer må kunne kjennes igjen i stor grad ved bruk av maskinlæring. Med stor grad mener vi i at dette bør være av en slik størrelsesorden at det merkbart vil lette arbeidspresset på analytiker.

Med hypotese 3 har vi i tankene at det kan oppstå problemer med at filteret vil klassifisere enkelte angrep som falsk alarm. Dette vil være uheldig da disse ikke vil gå til manuell analyse og dermed blitt oversett. Derfor vil vi undersøke om det finnes metoder innen maskinlæringsdelen som kan redusere dette problemet. Dette må ikke gå utover evnen til å oppdage falske alarmer i den grad at filteret mister sin hensikt.

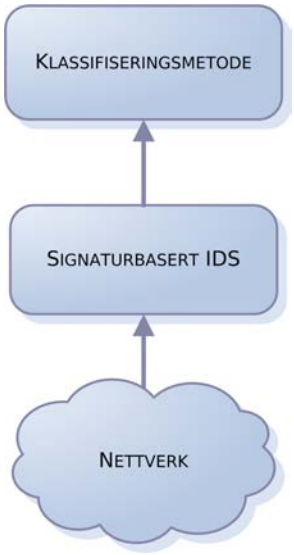
1.3 Oppgavebeskrivelse

Ordlyden fra oppgavebeskrivelsen ved innlevering 2.februar:

Oppgaven går ut på å forske på klassifiseringsteknologier til nytte innen «høyere ordens IDS», og velge en lovende læringsteknologi, som implementeres og testes ut.

Med «høyere ordens IDS» menes her en to lags IDS struktur, der andre lags IDS system siler alarmer fra første lags IDS, for å forbedre den totale ytelsen til IDS systemet, ved å senke raten av falske positive (falske alarmer).

Man ønsker også å sammenligne effektivitet til valgt teknologi med manuell/semi-automatisk logging samt drøfte fordeler og ulemper med automatisk logging.



1.4 Test case: Telenors innbruddsdeteksjonssystem

Gjennom denne oppgaven har vi samarbeidet med Telenor Sikkerhetssenters som tilbyr en nettverksovervåkingstjeneste i form av IDS. Her har vi

fått tilgang til de ressurser som treng for å gjennomføre testing av våre hypoteser. Telenors system består av det signaturbaserte innbruddsdeteksjonsystemet Snort, og et databasesystem der Snort-alarmer blir logget (lagret).

Dette gir oss tilgang til å benytte en stor mengde klassifiserte Snort-alarmer som kan nyttes til maskinlæringsprosessen. Dermed har vi en mulighet til å teste og sammenligne et slikt system med analyse-arbeid gjort manuelt.

Analysepersonellets oppgave er å klassifisere og behandle angrep som er oppdaget ved reelle alarmer. Stort sett vil behandling av angrepene innebære å rapportere disse til kunden i form av en fast rapport. Alvorlige angrep vil kunne kreve umiddelbare reaksjoner.

I databasen der Snort-alarmer er logget, er det reservert eget felt som benyttes til klassifisering. Her skal alarmene klassifiseres etter alvorlighetsgrad til angrepet, som kan angis i fem nivåer. Falske alarmer vil ikke bli klassifiser, noe som betyr at klassifiseringsfeltet for disse vil stå tomt. De fem nivåene for alvorlighetsgrad er definert som:

- Lovlig trafikk, verdt å legge merke til.
- Uønsket, men sannsynligvis ufarlig trafikk.
- Reelt angrepsforsøk, uten tegn på kompromittering.
- Vellykket angrep, eller helt nytt angrepsmønster som krever tiltak fra analytiker.
- Vellykket alvorlig angrep der kunde varsles øyeblikkelig.

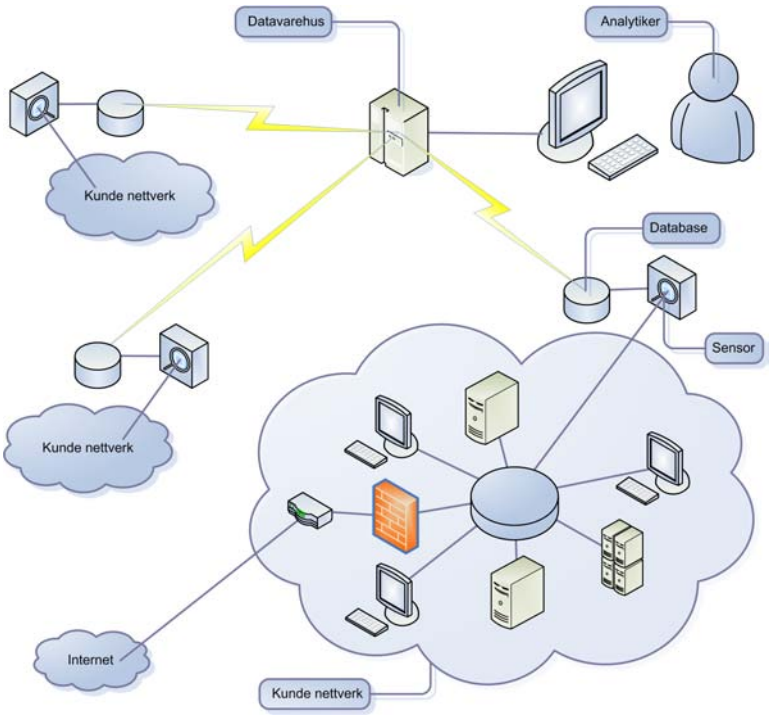
De to siste kategoriene vil gå litt over i hverandre, men det viktigste er uansett at rette aksjon blir gjort. Uansett grad av vellykkethet kreves det dyp analyse av analysepersonell.

Som ønske om å minske alarmer som går til manuell klassifisering, har Telenor et script som automatisk klassifiserer en god del av alarmene. Dette er alarmer som har trigger visse signaturer som skal behandles likt i alle tilfeller. Dette er et enkelt grep for å minske press på analytikere.

1.5 Beslektet arbeid og forskningsområder

1.5.1 Maskinlæring i IDS

Å benytte maskinlæring i IDS blir betraktet som et svært interresant felt innen forskning på nettverksikkerhet. Det som finnes av artikler og litteratur tar i hovedsak for seg muligheten for å benytte IDS som er maskinlært fullt



Figur 1.1: Telenors ids-system

ut. Et eksempel her å utvikle et større distribuert IDS basert på maskinlæring. Tanken her er å sette angriperen i fokus ved å lære mest mulig om oppførselen til vedkommende ved hjelp av informasjon fra flere nett [2]. Denne metoden vil kunne se sammenhenger i nettverkstrafikken og dra slutninger på bakgrunn av dette.

I artikkelen «netad» [6] ser man på anormalitetene som kan finnes i et angrep. Utfra teorier om at typiske feil og avvik vil forekomme i IP-stacken ved et angrep, kan dette utnyttes ved bruk av maskinlæring til å gjenkjenne disse trekkene. Forsøkene gjort i artikkelen «netad» er basert på eksperimentdata [3] der trafikken og angrep er simulert.

Et rent IDS basert på maskinlæring ser ikke på sammenhenger i dataflyten, men hver enkelt IP-datagram for seg selv. Likevel viser dette gode resultater totalt, man har et problem med at mange reelle angrep blir oversett i forhold til dagens signaturbaserte IDS.

Artikkelen «The neural network models for IDS . . . » [4] ser på bruken av «neural network» for å heve nøyaktigheten i et «data mining»-basert IDS. Bakgrunnen for prosjektet er at reelle alarmer som filtreres bort er mye farligere enn falske angrep som slipper igjennom. Nøyaktigheten kan derfor ikke kun baseres på hvor mange falske alarmer som fjernes, men må også ta antallet reelle alarmer som fjernes med i beregningen.

Ideen med å benytte ett signaturbasert IDS i «bunnen» med et maskin-

lært filter over dette («høyere ordens IDS»), for å sortere alarmer, gir oss muligheten til å kombinere det beste fra to verdener: signaturbasert IDS og maskinlæring, noe som vi ser på som veldig spennende.

1.5.2 Spamfiltre og den naive Bayesianske klassifikatoren

Maskinlæringsmetoden «naiv Bayes klassifikator», som vi skal ta for oss i detaljer senere, er idag implementert i forskjellige spam-filtre. Praksis viser at metoden har en god klassifiseringsrate når vi har tilstrekkelig treningseksempler til bruk i maskinlæring. Dette er noe en typisk får i en mailkonto, og i vårt tilfelle ved IDS-alarmer. Likheten mellom spam i mail og falske alarmer i IDS, er at en kan gjenkjenne mønstre og tydelig peke på enkelt elementer som vil bidra med klassifiseringen.

Et praktisk eksempel, nemlig spamfilteret «POPFile», er kort presentert i vedlegg D.

1.6 Rapportgjennomgang

Som en del av motivasjonen for innbruddsdeteksjon har vi i kapittel 2, datasikkerhet, en introduksjon til forskjellige angrepstyper og en forklaring av begrepet IDS. Vi gir en grundig beskrivelse av Snort, et signatur-basert IDS system vi kommer borti i en praktisk sammenheng under implementering og testing av vårt «høyere ordens IDS».

Kapittel 3, Maskinlæring, består av en generell introduksjon til begrepet maskinlæring fulgt av detaljert beskrivelse av de metoder vi vurderte å benytte.

Kapittel 4 argumenterer for hva som skal klassifiseres, og valg av maskinlæringsmetoden naiv Bayes klassifikator.

Framgangsmåte for å bygge opp et såkalt treningssett blir gjennomgått i kapittel 5. Her vil vi også ta opp spesielle problemer som dukket i forbindelse med vårt praktiske case.

Kapittel 6 ser på aspekter ved implementasjon av «høyere ordens IDS». Vi forklarer ikke implementasjonen i detalj men forklarer forskjellige metoder og funksjonaliteter som er utviklet for vårt «høyere ordens IDS».

Kapittel 7, presenterer de viktigste testene og resultatene fra arbeidet vårt. Resultatene her blir analysert og drøftet i påfølgende kapittel.

I kapittel 8 vil vi påpeke ulike forbedringer som kan gjøres for å øke ytelsen til filteret. Dette dreier seg både om vår implementasjon og loggerutiner. Vi påpeker også enkelte naturlige utvidelser av systemet.

Vi avslutter rapporten med å trekke konklusjon og liste av de viktigste referansene våre.

Kapittel 2

Datasikkerhet

En stadig økende mengde av skadelige angrep mot maskiner via Internet skaper behov for nettverksovervåkning i form av blant annet innbruddsdeteksjonsystem (forkortes IDS). I dette kapittelet vil vi utdype dette temaet generelt, og Snort IDS spesielt. Som nevnt i innledningskapitlet er Snort er et regelbasert IDS som vi jobber mot i våre forsøk, og vi vil derfor forklare dette grundig. Vi vil blant annet se på hvilke data som kan logges (lagres) fra alarmer. Dette er interresant da loggede data utgjør materialet som kan benyttes i den maskinlærte delen av et «høyere ordens IDS»

For å se litt på hensikten med å benytte innbruddsdeteksjonsystem på nettverk vil vi i slutten av dette kapitlet ta for oss ulike angrepstyper som viruslignende angrep og hacker-angrep. De ulike angrepsformene vil også diskuteres i forbindelse med signaturbaserte IDS.

2.1 Innbruddsdeteksjons-systemer

2.1.1 IDS-kategorier

Vi har idag to hovedtyper innbruddsdeteksjon-systemer, vertsbasert- og nettverksbasert ids.

Vertsbasert IDS består i å overvåke et system eller programloggfiler på jakt etter uønsket bruk av tjenester og ressurser. Nettverksbasert IDS, NIDS, er en teknologi som er utviklet med den hensikt å øke sikkerheten i et datanettverk. Ideen er å overvåke nettverkstrafikk for å oppdage uønsket trafikk. Dette kan innebære viruslignende angrep som sprer seg over et nettverk (ormer), eller hackerangrep. Skulle et IDS oppdage slik trafikk kan det reagere med å sende en alarm til nettverksansvarlig slik at vedkommende kan ta affære.

Vi vil konsentrere oss om NIDS videre, da det er dette som angår vårt arbeid.

2.1.2 Deteksjonsformer ved NIDS

Det finnes to hovedstrategier innen NIDS. Dette er «anomali-deteksjon» og «misbruks-deteksjon».

Ved «anomali-deteksjon» forsøker en å modellere normal nettverkstrafikk. Systemet utnytter denne kunnskapen til å reagere på nettverkstrafikk som avviker fra normalen. Styrken til denne innfallsvinkelen er at systemet teoretisk kan oppdage helt nye angrep uten å trenge oppdatering. Tanken her er at nye angrep også vil avvike fra det som betraktes som normal trafikk på nettverket.

Ved «misbruks-deteksjon» settes fokus på angrepene. Systemet er i dette tilfellet spesialisert til å gjenkjenne ulike former for angrep. Nye angrep kan derfor ikke oppdages før disse er observert og systemet er lært opp til å gjenkjenne disse. Det finnes flere teknikker innen både misbruk- og anomali-deteksjon [12].

Den mest brukte metoden innen IDS i dag er den såkalte signaturbaserte. Signaturbasert IDS utnytter at uønsket trafikk og angrep har ofte spesielle kjennetegn. Ved å lage regler (signaturer) basert på disse kjennetegnene skal systemet kunne fange opp angrep og ignorere lovlig trafikk. Signaturbasert IDS er derfor et eksempel på misbruks-deteksjon.

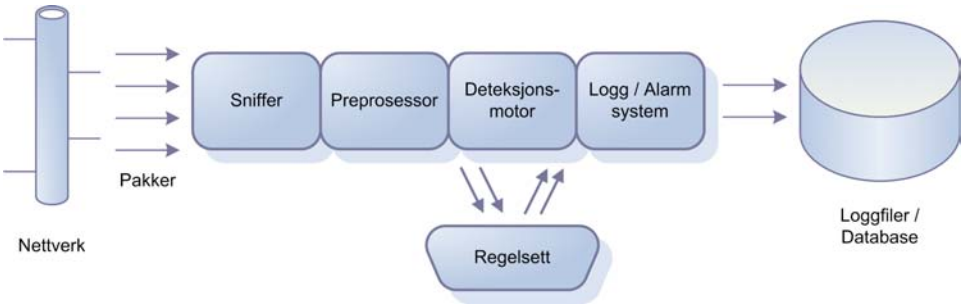
2.2 Snort

Snort er et signaturbasert IDS system som så dages lys helt på tampen av 1998. Det er et prosjekt basert på åpen kildekode og er derfor gratis for alle å ta i bruk. Snort er defor utbredt og har et omfattende basis regelsett (signaturer) som stadig oppdateres. Dette er viktig for et signaturbasert IDS da det i utgangspunktet ikke har noen mulighet å oppdage nye angrep før en signatur er skrevet. Vi skal ta for oss Snorts fire basis-komponenter:

- Sniffer
- Preprosessor
- Deteksjonsmotor
- Logg/Alarm system

2.2.1 Snifferen

Snifferens oppgave er å monitore nettverkstrafikken. Den vil analyserer pakkene, bestemme hvilke protokoller som brukes og gjøre dataene lesbare for mennesker. Dermed er de klare for videre behandling av Snort. I



Figur 2.1: Snort: funksjons-diagram. Fra kap 2.2.

de fleste tilfeller er det snakk om et IP nettverk, men Snort har også støtte en rekke andre nettverksteknologier som Ethernet, IPX, Apple Talk og PPP.

Det er vanlig å la snifferen overvåke nettverkstrafikken ved å la en switch speile all trafikken ut på en port mot snifferen som er satt til å fange opp denne trafikken[1]. Fordelen med et slikt oppsett er at IDS systemet ikke behøver en IP adresse for å motta trafikken den skal overvåke, og vil være usynlig for hackere utenfra.

2.2.2 Preprosessor

Snort har muligheter til å benytte såkalte preprosessorer. Dette er forskjellige plug-in, som benyttes på pakker før de blir sendt til deteksjonsmotoren. Preprosessorene kan benyttes etter ønske og behov, men flere av dem annbefales da disse har mye av æren for Snorts evne til å oppdage angrep. De forskjellige preprosessorene har svært ulik funksjonalitet, og vi vil nevne noen av de viktigste her.

- HTTP decode – Denne preprosessorens oppgave er å normalisere URI strenger til ASCII. Når denne er aktivisert vil en angriper ikke kunne lure signaturen ved for eksempel å benytte HEX isteden for ASCII i URI. I tillegg til HTTP, finnes det også preprosessorer for å normalisere Telnet, FTP og RPC trafikk. Disse heter henholdsvis Telnet decode, FTP decode og RPC decode.

- Frag2 – Defragmenterer trafikk for å gjenkjenne angrep som er spredt over flere datagram. Hindrer at hackeren kan lure Snort ved å fragmentere datagrammene som inngår i et angrep. Ulempen med dette er at viten om at et IDS som bruker frag2 kan utnyttes i ett DoS angrep. Ved å «spame» nettverket med mange fragmenterte pakker, vil IDS systemet bruke mye systemressurser på defragmentering.

[1] Faguttrykket for dette er å sette nettverkskortet i «promiscious mode».

- Portscan Detector – Det vil ikke la seg gjøre å skrive god en signatur på portscan siden hvert datagram vil fortone seg som en helt normal TCP oppkobling. Portscan-detektor vil derfor ikke se på en og en pakke men heller registrere om mange porter det forsøkes å kobles til innenfor en forholdsvis kort tidsperiode.

- Stream4 – Dette er en forholdsvis avansert preprosessor. Den sørger for at Snort holder orden på tilstanden til TCP forbindelser i trafikken. Dette gir ofte nyttig informasjon både for å gjenkjenne angrep og forhindre falske alarmer. Et eksempel på angrep stream4 oppdager er om en TCP forbindelse blir avsluttet uten at den noengang har eksistert (TCP-FIN pakke «out of state»). Dette er unormal oppførsel og kan bli benyttes i portscan siden serveren svarer med TCP-RST pakke om porten er stengt.

- SPADE – Dette er et forsøk på å innføre en mulighet for «anomaly detection» i Snort. Det vil si at denne komponenten er spesialisert til å gjenkjenne normal oppførsel (normal trafikk), og oppdage angrep ved avvikende oppførsel. SPADE er basert på *statistisk* anomalideteksjon.

2.2.3 Deteksjonsmotoren

Deteksjonsmotoren mottar pakker som er behandlet av preprosessorene for å sammenligne dem mot regelsettet (signaturene) i Snort. Om det finnes en eksakt match mot en regel sendes pakken videre til alarm/logg systemet. Signaturene er oppdelt i to nivåer:

- Rule Header – Her ligger informasjon om hvilkene protokoller regelen gjelder for, samt hvilke avsender- og destinasjons-IP-adresser som er interessante. Det vil også være angitt hva Snort skal gjøre dersom alarmen trigger, det vil i hovedsak si alarm og/eller logging

- Rule Options – Her står eksakt hva pakken skal inneholde for å trigge signaturen.

Eksempel på en enkel Snort signatur:

```
 alert tcp any 80 -> 192.168.1.0 any\
(content: "myhost"; msg: "myhost webside aksess";)
```

Første linje av signaturen er rule headeren, mens andre linje er rule options. Rule headeren består av følgende felt; Logg/alarm metode, protokoll, avsender-IP, avsender port, mottakers IP, mottakers port.

I eksempelets rule header er det valgt «alert» som logg/alarm metode. Dette indikerer både alarmring og logging av alarmen. Protokoll feltet sier vi kun er interresert i TCP trafikk. Avsenders IP er satt til «any», avsenders port «80». Dette betyr vi er interresert i allt som går på port 80 uansett IP avsenders IPadresse. Pilen (->) indikerer retningen på trafikken. Deretter kommer informasjon om mottakers IP og port. Denne rule headeren betyr altså vi er interessert i all tcp trafikk mot 192.168.1.0 som benytter port 80.

En signatur kan ha en rekke rule options. I eksempelet har vi to, «content» og «msg». Rule option «content» betyr at Snort vil lete gjennom payloaden etter det som er gitt her. I eksempelet vil altså alle pakker med «myhost» i payloaden få en eksakt match med oppgitte rule options. «Msg» angir ikke annet enn overskriften som skal vises ved logging og alarmering.

Signaturen i sin helhet vil føre til at forsøk på å browse websidene til «myhost» vil trigge alarm. Dette fordi webserveren «myhost» vil svare med et TCP datagram mot 192.168.1.0 på port 80, med «myhost» angitt i payloaden. Dette er selvfølgelig bare ment som ett eksempel, da dette ikke utgjør noe egentlig form for hacker angrep.

Rule options (Regelopsjoner)

I Snort finnes det femten opsjoner som kan benyttes. Det er ingen grense for hvor mange ganger hver av de kan benyttes i en signatur. Oversikt over opsjonene finner man i egen tabell 2.1 side 12.

2.2.4 Logg/Alarm system

Når en pakke matcher en signatur må Snort ha mulighet til å rapportere dette til en analytiker. Dette gjøres ved å sende alarm og logge informasjon rundt alarmen. Det er også mulighet for å kun logge alarmene.

Det finnes flere alternativer for logging og alarmering. Vi er spesielt interessert i logging, da særlig hvilken informasjon rundt alarmen som kan inngå her. Dette vil utgjøre materialet som senere kan benyttes til maskinlæring.

Informasjon som kan logges i Snort er fullstendig datapakke pluss informasjon fra som signatur, tidspunkt, signaturprioritet. Med fulstendig datapakke mener vi headerfelt fra transportprotokoll (TCP/UDP/ICMP) og hederfelt og payload fra Internet protokollen. Det finnes ulike alternativer for hvor mye av dette som er ønskelig å logge. I vår case var derimot fullstendig loggin benyttet slik at all denne informasjonen var tilgjengelig. Fullstendig logging er ønskelig ved et «høyere ordens IDS», fordi dette gir den maskinlærte delen av systemet mer data til bruk i læringsprosessen.

Ulike formater kan benyttes til logging. Feks XML, tekstfil eller database. En velge om man vil logge dataene binært, hex eller i ascii. For maskin-

msg	skriver en beskjed i alarmen og loggen
logto	logger pakken i en brukerspesifikk fil istedenfor standard loggfil
minfrag	setter en grense for den minste akseptable størrelsen p etå IP-fragment
ttl	tester IP headerens TTL felt
id	tester header feltet IP-ID for en spesifikk verdi
dsize	tester pakkens payload størrelse
content	leiter etter oppgitt mønster i pakkens payload
offset	modifiserer content opsjonen, setter offset til å foreta mønstersøk i payload
depth	modifiserer content opsjonen, setter maks dybde for payloadsøk
flags	tester TCP flagg
seq	tester TCP sekvensnummer feltet
ack	tester TCP-ACK feltet
itype	tester ICMP type feltet
icode	test ICMP code feltet
session	dumper informasjon fra applikasjonslaget for en gitt sesjon

Tabell 2.1: Regelopsjoner i Snort

læringens del spille dette ikke noen rolle. Ascii kan være greit slik at mennesker kan forstå dataene.

Som et eksempel ser vi på en logget alarm som er generert ved å browse nettsiden «myhost» når eksempelsignaturen vår benyttes. Vi viser fullsendig logging.

```
[**] [1:0:0] myhost webside aksess [**]
04/14-19:44:54.571931 216.109.125.72:80 -> 192.168.1.1:32802
TCP TTL:46 TOS:0x0 ID:51401 IpLen:20 DgmLen:1492 DF
***A**** Seq: 0x668B0B47 Ack: 0x833208B7 Win: 0x8160 TcpLen: 32
TCP Options (3) => NOP NOP TS: 232478239 85744

* Her vil payloaden ligge *
```

Som vi ser inneholder alarmen informasjon fra pakkenheaderene pluss payload (som vi ungår å liste opp). Den matcher signaturen vår da den inneholder ordet «myhost» i payloaden [2] og den består av en TCP pakke som sendes på port 80 mot IP-adressen som var angitt. Rule options "MSG" ser vi igjen som en slags overskrift for alarmen.

2.3 Angrepstyper

I denne seksjonen presenterer vi ulike typer angrepsformer og hvilke skader disse kan medføre. Angrep stammer fra personer som ønsker å stjele og/eller ødelegge data, bevise sin posisjon som hacker eller for spenningens skyld. Vi omtaler den som utfører angrep som en hacker.

Motivet til hackeren vil i stor grad avgjøre hvor stor skade de gjør. En hacker som kun er ute etter spenning vil som regel ikke å volde offeret noen skade. Men selv «harmløse» angrep kan føre til store økonomiske tap. Bedrifter eller organisasjoner som oppdager at hackere har vært og snoket, har i flere tilfeller sett seg nødt til å omprogrammere store mengder kildekode.

Et nylig eksempel på en slike hendelse er tegn som tydet på et vellykket hacker-angrep på serveren «gnome.org». Dette angrepet lot seg ikke kartlegge skikkelig, og det ble derfor spekulert i om noen hadde lagt inn et sikkerhetshull i kildekoden til den nye versjonen av Gnome [3]. Dette resulterte i en omfattende gjennomgang av koden.

En kan i hovedsak skille mellom to ulike angrepstyper. Et angrep som direkte involverer en hacker kaller vi for hackerangrep. Dreier det seg om små program eller script skrevet av en hacker som utfører angrepet automatisk kaller vi dette for virus-lignende angrep. Det finnes en rekke ulike varianter av begge typer.

2.3.1 Viruslignende angrep

Det vi har valgt å kalle viruslignende angrep er den vanligste formen for angrep på Internet. Grunnen er at disse har evnen til å spre seg selv, i stort omfang, uten hjelp fra en hacker. De ulike viruslignende angrep har ofte en ganske klar karakteristikk, som gjør dem enkle å oppdage. Vi vil se på ulike varianter av viruslignende angrep her.

Datavirus

Datavirus har eksistert i en årrekke. Dette er små program som automatisk legger seg til andre filer. Viruset har den evnen at det kan kopiere seg

[2] Dette kan ikke leses her, men er grunnen til at det er generert alarm

[3] Siden Gnome 2.6 var under utvikling på denne tiden

selv for å infisere flere filer i datamaskinen. Spredning til nye vertsmaskiner skjer ved filoverføring. Skadeeffekten av virus kan variere fra en humoristisk beskjed fra hackeren, til å ha svært ødeleggende effekt på systemet.

Ormer

Ormer ligner datavirus, men har en evne til å automatisk spre seg til nye maskiner. Dette uten å være avhengig av filoverføring eller annen aktivitet basert på brukeren av den infiserte maskinen. Automatisk generering av epost er en typisk måte en orm kan spre seg på. Det har vist seg at enkelte ormer kan sprer seg utrulig raskt, og kan i løpet av få timer infisere datamaskiner over store deler av kloden. «Nimda» [9] ormen som oppstod september 2001 er kanskje det mest kjente eksempelet vi kan nevne på en slik orm. Ormer med rask spredning kan skape problemer for regelbaserte IDS, da nettverket kan bli angrepet før en signatur for ormen er skrevet.

Trojansk hest

Navnet er hentet fra gresk mytologi. En trojansk hest opptrer som et tilsynelatende ufarlige program med kjent funksjonalitet, som en skjermbeskytter eller ett spill. Ved installasjon vil imidlertid en bi-funksjon i form av et virus eller rootkit-funksjon bli aktivisert på maskinen.

Root kit

Root kit vil åpne en port der en hacker kan kommunisere og ta full kontroll over maskinen. I dette tilfellet kan en hacker få større kontroll over systemet enn offeret har selv. Root kit kan som nevnt følge med en trojanske hest, slik at hackere gjerne må lete seg fram til de maskiner som er infisert. Som et eksempel på en root kit kan vi nevne «SubSeven». Denne gjorde hackeren i stand til å bevege musen, skru av monitoren, og aktivere et eventuelt webkamera for å kikke på brukeren uten at vedkommende er klar over det.

Hybrider

Hybrider brukes som en betegnelse når man støter på noe som kombinerer flere av oppførslene nevnt over.

2.3.2 Hackerangrep

Med hacker angrep menes at en person utfører et målrettet angrep mot et nettverk eller en datamaskin. Det finnes mange former for slike angrep,

og enda flere framgangsmåter. Vi skal her se på noen eksempler på ulike former for hacker angrep, og typiske framgangsmåter for disse.

Footprinting

I forkant av et angrep benytter hackere ulike metoder for å finne et mål og å kartlegge hull i sikkerheten han kan utnytte for å få tilgang til systemet. Slik aktivitet kalles for Footprinting. Metodene her er mange, og det finnes ofte flere veier til samme mål.

I vårt tilfelle er metodene som berører nettverket de mest interessante, dette kan være ting som port skanning eller skanning etter Root Kit. De vanligste metodene for å kartlegge nettverk er å sende multiple forespørsler itl et utvalg addresser på interessante porter, eller generell icmp-request-meldinger [4]. Disse metodene går under kategori av forskjellige typer «scan». Målet er å kartlegge svakheter som kan utnyttes i et angrep. «Scan» som ikke kan oppdages uten at man ser på sammenhengen i trafikken vil det ikke være mulig å skrive en signatur på. F.eks må Snort benytte en egen «prekompressor» for å oppdage portscann.

IP-sniffing er an annen form for footprinting. Her brukes avlyttes nettverket for å kartlegge IP-adresser, tjenester og servere. Kunskap som tilegnes på denne måten kan senere utnyttes i et angrep. Det kan også være mulig å snappe opp mer kritisk informasjon, som ukrypterte passord, e-post som inneholder sensitiv informasjon, kredittkort nummer, etc. Slik aktivitet er vanskelig å oppdage ved alle former for IDS da hackeren ikke genererer noe trafikk mot nettverket.

System hacking

Et forsøk på system hacking vil arte seg helt forskjellig fra system til system. Et sikkerhetshull som eksisterer i Windows 2003 finnes ikke nødvendigvis på et system som kjører Linux eller omvendt. Et slikt angrep krever derfor en del kunnskap om plattformen systemet som angripes kjører på. Dette er gjerne aktivitet som kommer i etterkant av footprinting. System hacking som utnytter sikkerhetshull vil ofte ha typiske trekk som gjør det greit å skrive signaturer på dem. Typisk trekk kan være spesielle porter, eller fast mønster i payload.

DoS - Denial of Service

DoS angrep går ut på å overbelaste offeret med massive mengder data eller forespørsler. Dette vil si å oversvømme offeret med datapakker. Disse

[4] ICMP-request og response, er kanskje bedre kjent som forespørsler og svar som det velkjente verktøyet «Ping» bruker.

kan dreie seg om vanlige ICMP ping pakker, eller andre datapakker uten nyttig innhold.

Ved et klassisk DoS-angrep vil angriperen forsøke å overbelaste programvaren, for å ta ned en server eller lignende. Distribuert DoS (DDoS) angrep er tilfellet der angriperen har som mål å sette nettverket til offeret ut ved å oversvømme dette med unyttig trafikk.

Å spore angriperen til et DoS/DDoS angrep er ofte vanskelig da vedkommende gjerne trekke en rekke «uskyldige» parter med i angrepet. Dette kan for eksempel gjøres ved spredning av en orm.

DoS-angrep er et problem som er vanskelig finne gode tiltak mot. Et IDS kan settes ut av spill ved å sender et DoS angrep mot nettverket med pakker som vil generere alarmer av IDS, som f.eks portscan. Dette vil fører til at IDS systemet oversvømmes av alarmer, og kan dermed settes ut av spill slik at et mer alvorlig angrep kan settes i verk uten å bli oppdaget.

Remote control

Mange servere kjører en eller annen form for programvare for å kunne administrere maskinen uten å være fysisk tilstede. Faren her er at det kan dukke opp svakheter som gjør en hacker i stand til å ta kontroll over serveren. Årsakene kan være dårlig sikkerhet eller feil i programmet som tilbyr «remote controll», kombinert med dårlig nettverksikkerhet generelt. Sendes f.eks brukernavn og passord ukryptert på nettverket kan dette fanges opp ved IP-sniffing, og benyttes i et remote controll angrep.

Denne angrepsformen kan enkelt oppdages ved å skrive en signatur som reagerer på aktiviteter relatert til pålogging o.l fra fremmede IP-adresser.

Web (server) hacking

Som tidligere nevnt inneholder ofte (server-)programvare feil som gjør det mulig å få den til å gjøre ting den normalt ikke skulle gjort, som for eksempel å kjøre kommandoer, endre innstillinger eller ta i mot filer som endrer på websiden serveren publiserer (på engelsk kalt DeFacing). Et mer alvorlig angrep kan begynne på samme måten som et DeFacing-angrep, men i steden for å endre på websiden laster han opp og kjører programmer. Disse kan deretter brukes til å ta fullstendig over kontrollen av serveren.

Kapittel 3

Maskinlæring

I dette kapitlet vil vi gå inn på temaet maskinlæring. Maskinlæring består av et sett metoder som muliggjør at datamaskiner selv kan ta for seg opplæring for å kunne løse en gitt oppgave.

I starten av kapitlet snakker vi om generelle aspekter rundt maskinlæring, for så å forklarer prinsippene for tre spesifikke maskinlæringsalgoritmer som alle ble vurdert i henhold til denne oppgaven. Disse tre baseres på ulike prinsipper innen maskinlæring. Algoritmene dette gjelder er:

- Beslutnings-tre
- k-Nærmeste nabo
- Naiv Bayes klassifikator

3.1 Introduksjon til maskinlæring

Maskinlæring er et felt innen automatiserte systemer og kunstig intelligens som benyttes i økende grad. Vi finner eksempler på det i datamining-oppgaver, for eksempel ved å finne *trender* i store pasient databaser som benyttes som hjelp til å stille diagnoser. Det er også utviklet program basert på maskinlæring for å spille spill som sjakk og backgammon, gjenkjenne håndskrevne tekst i bilder, eller gjenkjenne tale, for å nevne noe. Følgende definisjon på maskinlæring er hentet fra Tom M. Mitchells bok om maskinlæring [8].

> Et dataprogram kan sies å lære fra erfaring E med hensyn til en klasse av oppgaver O og ytelse Y, hvis ytelsen på oppgavene i O målt med Y, øker med erfaringen E.

Maskinlæring går altså ut på å la et program selv kunne lære seg å utføre en oppgave ved å bli trent opp. Hva oppgaven og treningen består

i kan variere. Om oppgaven er å klassifisere Snort-alarmer kan treningen bestå av en mengde ferdig klassifiserte Snort-alarmer som maskinlæringsalgoritmen går igjennom. Algoritmen må fungere slik at den automatisk finner sammenhenger i data fra de ferdig-klassifiserte alarmene, siden dette brukes for å klassifisere nye alarmer. Ser vi på dette som et eksempel i sammenheng med definisjonen over har vi følgende:

Eksempel. *Sortering av Snort-alarmer.*

- *O oppgave - Klassifisere alarmer.*
- *E erfaring - Ferdig klassifiserte alarmer.*
- *Y ytelse - Andel riktig klassifiserte alarmer.*

Oppgaven vil som regel gi seg selv. Ønsker man for eksempel et sjakkprogram, vil oppgaven bestå i å spille sjakk. Likevel er det viktig at oppgaven er entydig definert.

Erfaring kan høstes på ulike måter etter hva oppgaven består av. I eksemplet over med Snort-alarmer, besto treningen av å presentere en mengde data med en fasit (treningseksempler). Dette kalles direkte læring. Ved direkte læring er algoritmen avhengig av en læremester som presenterer eksempler med «fasit», for å øke erfaringen. Læremesteren i eksempelet over er vedkommende som har klassifisert alarmene som benyttes til læring. Eksemplene som inngår i direkte læring kalles treningseksempler.

Indirekte læring er tilfeller der ingen læremester trengs for at programmet skal tilegne seg erfaring. En algoritme som spiller dam kan lære indirekte ved å spille partier mot seg selv. For å øke spillestyrke må da algoritmen bruke resultatet av partiet og trekk rekkefølgen til læring. Alternativt kan det benyttes direkte læring også her, ved å gi algoritmen en rekke treningseksempler på stillinger, med tilhørende beste trekk (fasit).

Ytelse er et mål på hvor bra en oppgave løses. Hvordan dette måles og hva som er tilfredsstillende resultat kommer helt an på oppgaven og algoritmen som er benyttet. Ytelsen bør ikke måles på treningseksemplene, da resultatet da kan bære preg av spesialisering på disse.

3.1.1 Forskjeller mellom menneske og maskin

Mennesket og datamaskin har svært ulike måter å resonnere på, noe et spill som sjakk kan illustrere. Mennesket har ingen mulighet å henge med på hastigheten en datamaskin regner trekk i en sjakkstilling. En av de store sjakkmaskinene i dag «regner» hundrevis av millioner trekk i sekundet. Dette er flere trekk enn hva en sjakkmester «regner» iløpet av hele livet. Likevel opplever vi at sjakkmaskinene ofte har problemer mot verdens toppspillere i sjakk.

En forklaring på hvordan dette er mulig er menneskets intuisjon. En sjakkspiller kan øyeblikkelig gjenkjenne gode og dårlige trekk i en stilling uten å behøve å «regne» på dem. Det betyr at sjakkspilleren sparer mye tid ved å utelukke de aller fleste trekkmulighetene når de skal beregne beste trekk i en stilling. Dette gjør at de kan konkurrere mot datamaskinens overlegne hastighet. Til dags dato er ikke mulig å implementere denne intuisjonen i noen form for dataprogram. Datamaskinen må derfor basere seg av andre prinsipper når den spiller sjakk, noe som kan fortone seg kryptisk for et menneske. Her skal vi se på et eksempel.

En algoritme for å spille sjakk kan bestå av en mengde parametere som representerer ulike fordeler og ulemper i en gitt stilling. Enkle eksempler er hvor mange brikker hver spiller har av hver type, hvem har rokert, hvor mange brikker står i slag og så videre. Algoritmen kan utfra disse parametrene bedømme hvor god en gitt sjakkoppstilling er. Ved å regne ut dette for alle stillinger som kan oppstå ved alle trekkmuligheter vil det beste trekket kunne bestemmes. I «Machine Learning» [8] er det gitt eksempel på et program som spiller dam der logikken på denne typen algoritme blir gjennom gått grundig.

Skal algoritmen fungere må de ulike fordeler og ulemper vektes riktig i forhold til hverandre. Det er denne vektingen som er vanskelig å gjøre riktig for et menneske, da vi tenker på sjakk på en helt annen måte. Dette er det nettopp hva som gjøres automatisk ved maskinlæring. Ved å tilføre erfaring, enten at maskinen spiller med seg selv eller ved produserte treningseksempler, kan vektingen av parametrene automatisk bli bestemt av datamaskinen. Jo mer erfaring jo bedre vil vektingen bli. Dermed vil ytelsen (spillestyrke) øke på å utføre oppgaven (spille sjakk).

Problemet med at mennesker og maskiner tenker på svært ulike måter kan derfor, til en viss grad, sies å løses ved maskinlæring. Riktignok må vi implementere læringsalgoritmen, men deretter er det dataprogrammet selv som lære seg opp slik at programmet har den type informasjon som skal til for å løse sin oppgave.

3.1.2 Fordelene med maskinlæring

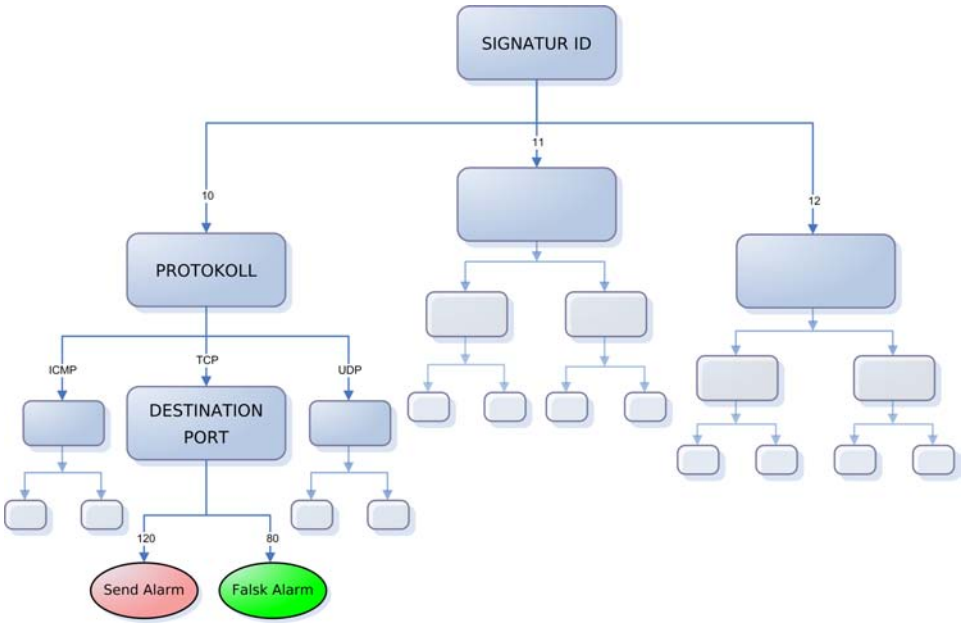
Fordelen med maskinlæring er at programmet tar hånd om opplæringen selv. Hverken programmerer eller bruker trenger vite noe om de faktiske «reglene» algoritmen utvikler for å løse sin oppgave. Datamaskinens hurtighet gjør det også mulig for å benytte svært store datamengder til læring. Maskinlæring er derfor nyttig når man ønsker et program som kan løse kompliserte oppgaver (spille sjakk) eller for å finne trender fra store mengder med klassifisert data (sortere Snort-alarmer). Maskinlæring er også svært anvendelig i programmer som er avhengig av å kunne endre karakter over tid. Dette kan gjøres ved kontinuerlig trening, eller ved å trenes opp på nytt når behovet melder seg.

3.2 Besluttningstre

Beslutningstre er en mye brukt maskinlæringsalgoritme. Den sorterer treningseksempler utover i en trestruktur. Treet består av en rekke noder som forgrener seg i nye noder. Disse nodene representerer ulike attributter fra treningseksemplene. Attributter er de enkeltelementene treningseksemplene er bygget opp av. Snort-alarmer er bygget opp av headerfelt til ulike datagrammer, mens pasientjournaler av personlig informasjon som kjønn og alder, som to eksempler.

Attributtverdier som er angitt i en node bestemmer hvilken forgrening man skal følge i treet. De ytterste bladene på treet vil inneholde klassifiseringsdata. Ved å følge et beslutningstre fra «rot» til «blad» vil man så tilslutt ha klassifisert en instans.

Her er et eksempel på et lite utsnitt av et besluttningstre for å klassifisere Snort-alarmer. Rot-noden er den Snort-signatur som er trigget. De andre nodene er ulike elementer fra headeren til den pakken som trigget alarmen.



Figur 3.1: Beslutningstre

Får vi en alarm der SIGNATUR ID = 10, PROTOKOLL = TCP og DESTINATION PORT = 80 kan vi følge treet fra «rot» til «blad», og dermed følge hvordan klassifiseringen logisk sett foregår. Denne alarmen vil klassifiseres som falsk i dette tilfellet.

3.2.1 Læringsalgoritmen

Det finnes flere ulike algoritmer for å bygge ett beslutningstre. Den grunnleggende algoritmen heter ID3. Den finnes i flere utvidede versjoner for ta hånd om problemer som overtilpassing og liknende. C4.5 er et eksempel på en mye brukt algoritme som er en slik utvidelse av ID3.

Hovedstrategien når man skal bygge et beslutningstre er å ende opp med et minst mulig tre, som likevel gir maksimal ytelse. Et lite tre som klassifiserer like godt som et større ansees som bedre utfra argumenter som ytelse og effektivitet. For å ende opp med et lite og effektivt tre er det viktig at de attributtene som klassifiserer treningseksemplene best benyttes lengst oppe i treet. ID3 forsøker å gjøre dette ved å regne ut en verdi for reduksjonen i entropien til treningseksemplene, ved hver attributt om disse blir benyttet som «rot». Denne verdien kalles for attributtets bidrag (eng «information gain»). Attributten med høyest bidrag settes som rot i treet, og prosessen gjentas for å avgjøre hvilke attributter som bør utgjøre neste nivå i treet. Slik fortsetter algoritmen til den er kommet igjennom alle treningseksemplene.

Med entropien til et sett med klassifiserte data, menes en verdi som gir en størrelse på hvor stor spredning disse dataene er med hensyn på klassen. Dette kan uttrykkes matemetisk på følgende måte:

$$Entropy(S) = \sum_{i=1}^{h} -P_i \log^2 P_i \tag{3.1}$$

der
P_i er sannsynligheten for hypotesen i,
S er et sett med treningseksempler, og
h er antall hypoteser.

Bidrag beregnes deretter ut fra Entropien på følgende måte:

$$Gain\,(S, A) = Entropy\,(S) - \sum_{v \epsilon Verdier(A)} \frac{|S_v|}{|S|} Entropy\,(S_v) \tag{3.2}$$

der
S er et sett med treningseksempler, og
A er attributt A.

3.2.2 Forbedring av metoden

Det kan oppstå problemer med læring basert på besluttningstre dersom treningseksemplene inneholder mye støy (feil). En ende da opp med et tre som er spesialtilpasset treningseksemplene. Dette kalles overtilpassing,

og betyr at treet har fortsatt å forgrene seg utover det punktet der treet er optimalt. Dette skjer fordi læringsmetoden vil forsøke å lage et tree som klassifiserer treningseksemplene helt perfekt. Dermed vil treet utvides med alle støyende (feilklassifiserte) treningseksempler. Et overtilpasset tre vil yte dårligere på testeksempler utenom de som er benyttet i læringen.

For at algoritmen skal tåle støy er det derfor nødvendig å forhindre overtilpassing. Det finnes to strategier for dette. Treet kan stoppes å utvikles før det nå et stadium der det blir overtilpasset, eller treet kan overtilpasses for så etterpå kutte treet til det punktet det yter maksimalt (post-prune).

3.3 K-Nærmeste Nabo

Ved bruk av nærmeste nabo som metode for læring ligger den komplekse biten i klassifiseringsalgoritmen. Trening foregår enkelt ved at treningseksemplene lagres i en liste. Hvert treningseksempel ansees som et punkt i rommet R^n, der n er antall attributter. Verdien av hvert punkt angir posisjonen til punktet i R^n. Når en ny instans skal klassifiseres betraktes også dette som ett punkt i R^n, lokalisert etter attributtverdiene. Instansene blir klassifisert utifra de K nærmeste punktene fra treningseksemplene. Matematisk kan algoritmen presenteres slik:

Gitt en instans x_v som skal klassifiseres

$$\hat{f}(x_q) \leftarrow \underset{v \epsilon V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i)) \tag{3.3}$$

hvor $x_1, \ldots, x_n$ er k instanser av treningseksempler nærmest til x_q

Avstanden mellom to punkt p og q i R^n, der p er gitt ved vektoren $(p_1, p_2, p_3, \ldots, p_n)$ og q ved $(q_1, q_2, q_3, \ldots, q_n)$:

$$d(p, q) = \sqrt{\sum_{r=0}^{n} (p_r - q_r)^2} \tag{3.4}$$

For å gi et eksempel kan vi anta vi ønsker å klassifisere en instans der vi har valgt $K = 5$ algoritmen. La oss si vi har to klasser, «pos» og «neg». Algoritmen vil betrakte de 5 punktene fra treningseksemplene som ligger nærmest punktet til instansen som skal klassifiseres. Fire av disse er klassifisert som «pos» og en som «neg». Resultatet av klassifiseringen ville da blitt «pos». Om vi derimot hadde satt $K = 1$, og punktet «neg» var den aller nærmeste av disse fem, ville klassifiseringen landet på «neg».

3.3.1 Forbedring av metoden

Klassifiseringsalgoritmen kan utvides med å avstands-vekte punktene. Det vil si at punkter fra treningseksemplene som ligger nærme instansen som skal klassifiseres teller mer enn de som ligger lengre unna. Vektingen bruker å settes til en over kvadratet av avstanden. Skulle det oppstå an situasjon der avstanden er null, lar man dette punktet alene bestemme klassen til instansen.

Et problem med K-nærmest nabo er at alle attributter i utgangspunktet vil telle like mye ved klassifiseringen. Dette fører til at irrelevante attributter vil kunne gjøre at to punkt får stor avstand og dermed føre til instansen blir klassifisert feil. For å bøte på dette kan man vekte hver attributt ulikt, slik at viktige attributter teller mer. Denne vektingen blir bestemt ved testing (kryssvalidering), og er en forholdsvis tung prosess om det er snakk om mange attributter. Alternativt kan også dårlige attributter fjernes helt.

3.4 Naiv Bayes klassifikator

Naiv Bayes klassifikator er en algoritme basert på sannsynlighetsregning som har sitt utspring i Bayes teorem. Algoritmen er egnet når vi har treningseksempler som inneholder et gitt sett med attributter, og vi har et endelig sett med klasser. Naiv Bayes er både enkel å implementere og viser seg dessuten å gi svært gode resultater. Det første vi vil se på er Bayes teorem, som den naive Bayes klassifikator bygger på.

3.4.1 Bayes teorem

Vi har et sett med hypoteser H og observert data D. Følgende notasjon benyttes:

- $P(h)$ - Sannsynligheten til hypotese $h \epsilon H$.
- $p(D)$ - Sannsynligheten til treningseksemplene D (uavhengig hvilken hypotese).
- $p(D|h)$ - Sannsynligheten for treningseksemplene D gitt hypotesen h.
- $P(h|D)$ - Sannsynligheten for en hypotese h utfra treningseksemplene D.

Bayes Teorem:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{3.5}$$

Teoremet gir mulighet til å regne ut sannsynligheten til en gitt hypotese, basert på forkunnskaper om sannsynlighetene $P(h)$ og $P(D)$ samt et sett

med treningseksempler D. Det vi generelt er ute etter i alle former for maskinlæring er den mest sannsynlige hypotesen. Vi kaller den MAKS hypotesen. Om H består av et endelig antall elementer har vi følgende formel for h_{MAKS}

$$\begin{aligned} h_{MAKS} &= \underset{h \epsilon H}{\operatorname{argmax}} P(h|D) \\ &\Downarrow \\ h_{MAKS} &= \underset{h \epsilon H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)} \end{aligned} \tag{3.6}$$

$P(D)$ fjernes siden den utgjør en konstant og ikke vil påvirke hvilken hypotese som kommer ut med den største sannsynligheten. Vi får da:

$$h_{MAKS} = \underset{h \epsilon H}{\operatorname{argmax}} P(D|h)P(h) \tag{3.7}$$

3.4.2 Klassifikatoren

For å benytte naiv Bayes klassifikator, forutsettes et sett treningseksempler der hver instans består av en rekke attributter og en endelig mengde klasser (hypoteser) K der $k \epsilon K$.

Hver instans kan skrives på følgende form: $(a_1, a_2, \ldots, a_n)$, der a_n representerer verdiene til attributtene. vi har da:

$$k_{MAKS} = \underset{k_i \epsilon K}{\operatorname{argmax}} P(k_i|a_1, a_2, \ldots, a_n) \tag{3.8}$$

Ved å benytte Bayes teorem kan vi uttrykke k_{MAKS} slik:

$$\begin{aligned} k_{MAKS} &= \underset{k_i \epsilon K}{\operatorname{argmax}} \frac{P(a_1, a_2, \ldots, a_n|k_i)P(k_i)}{P(a_1, a_2, \ldots, a_n)} \\ &\Downarrow \\ k_{MAKS} &= \underset{k_i \epsilon V}{\operatorname{argmax}} P(a_1, a_2, \ldots, a_n|k_i)P(k_i) \end{aligned} \tag{3.9}$$

Ut fra formel 3.9 vil altså sannsynligheten for at en instans tillhører en gitt klasse bli bestemt utfra sannsynligheten for denne klassen multiplisert med sannsynligheten for instansen, gitt klassen. Klassifiseringen ender opp med den klassen med høyest sannsynlighet.

Dette kan by på problemer om instansene inneholder mange attributter. Grunnen er at hver av disse attributtene innehar attributtverdier som kan variere. Om vi ikke har svært mange treningseksempler tilgjengelig, kan det derfor være sjeldent flere instanser er like. Da vil det også være vanskelig å gi noen god statistikk på hvor ofte instanser forekommer for de

ulike klassene. I slike tilfeller vil det finne en rekke enkeltstående instanser blant treningseksemplene, og vil ikke kunne hjelpe klassifiseringen.

I metoden Naiv Bayes klassifikator gjøres noe med dette problemet med å anta at alle attributtene er totalt uavhengig av hverandre. Dermed sier man at sannsynligheten for hele instansen gitt klasse er lik produktet av sannsynligheten for hver enkelt attributtverdi gitt klasse. Dette er en forenkling som gjør at læringsmetoden ikke er avhengig av ett enormt treningssett for å fungere. Det viser seg også at denne tilnærmingen i mange tilfeller fungerer svært godt. Dette gir følgende formel som definerer naiv Bayes klassifikator.

$$k_{NB} = \underset{k_j \epsilon K}{\operatorname{argmax}} P(k_j) \prod_i P(a_i|k_j) \tag{3.10}$$

I tilfeller der det er vanskelig å gi noe godt mål på sannsynligheten for hver klasse settes $P(k_j) = 1$

Et problem oppstår når en attributtverdi ikke er representert for en eller flere av klassene i treningseksemplene. Formelen 3.10 vil da gi sannsynlighet for verdien, gitt klassen, lik null. Den endelige sannsynligheten for klassen som er et produkt av alle attributtverdienes sannsynligheter, vil derfor også ende opp som null. Med andre ord vil dette veie urealistisk mye.

For å benytte naiv Bayes klassifikator er man avhengig å ha det statistiske materialet som benyttes i formel 3.10 tilgjengelig. Det får man ved rett og slett en opptelling av attributtverdier og klasser fra treningseksemplene. Denne prosessen kan sees på som selve læringsprosessen. Dette materialet må så struktureres på en slik måte at det er enkelt å benytte av klassifisatoren.

Antagelsen om at attributtverdiene er uavhengig hverandre vil i få tilfeller stemme. Flere attributter blir gjerne påvirket av felles kilder, slik at de vil ha en liten avhengighet. Dette betyr at flere attributter vil gi litt av den samme informasjonen (den som kommer fra felles kilder). I bayesisk nett, som er en annen maskinlæringsmetode basert på Bayes metode, blir dette tatt hensyn til ved å vekte de ulike attributtene.

Naiv Bayes klassifikator benytter attributtene direkte uten vekting. Dette vil føre til høyere sannsynligheter for attributtverdier enn hva som faktisk skulle vært tilfelle. Dermed vil vi få mer ekstreme sannsynlighetsforskjeller mellom klassene. At metoden likevel viser seg å fungere svært godt, kommer av at ved klassifisering trenger en kun å vite den mest sannsynlige klassen, som vi fortsatt er i stand til å regne ut.

3.5 Maskinlæring og IDS

Vi ønsker å bygge et filter som sorterer Snort-alarmer ved benytte maskinlæring. Hovedoppgaven vil være å skille falske alarmer fra alarmer som

indikerer virkelige angrep (reelle alarmer). Ved vårt samarbeid med Telenor sikkerhetssenter har vi tilgang til en mengde ferdig klassifiserte Snort-alarmer, fra en rekke ulike nettverk. Disse vil bli benyttet som treningseksempler til direkte læring av den naive Bayesiske klassifikator.

Som nevnt i kapittel 2, kan snort logge (lagre) data som ligger i headerfelt fra internet-protokoller (her menes IP og TCP/UDP/ICMP) og payload fra Internet-protokollen fra pakkene som utløser alarmer. Disse omtales videre som attributter. For at systemet vi utvikler skal være direkte portabelt mot andre regelbaserte IDS, må vi ta et forbehold at disse har mulighet til å logge mange av de samme attributtene.

For at maskinlæring som metode skal fungere, er vi avhengig av at det finnes små statistiske forskjeller på hvor sannsynlig ulike attributtverdiene er for hver klasse. Klassifisering skjer på grunnlag av en samlet «vurdering» av hva attributtverdiene antyder. Vi vil her gi noen eksempler elementer i sattributtene vi mener vil kunne utnyttes ved maskinlæring på Snort-alarmer. Dette gjelder først og fremst reelle alarmer kontra falske alarmer.

- Det kan oppstå atypiskheter eller feil i headerfeltene til internet-protokollene ved angrep. Dette kan være en følge av feil eller slurvete implementasjon fra hackeren.
- Enkelte hacker-verktøy vil kunne kjennes igjen ved spesielle verdier i headerfeltene til internet-protokollene. For eksempel kan et slikt alltid benytte samme portnummer og liknende.
- Angrep kan ofte kjennes igjen ved typiske trekk i payloaden til IP segmentet.
- Signatur vil kunne gi en god sannsynlighet på angrep eller ikke, da noen signaturer nesten alltid er falsk alarm og motsatt at noen signaturer uten unntak regnes som reelt angrep.

Kapittel 4

Klassifisering av IDS-alarmer

Vi har i de foregående kapitlene tatt for oss datasikkerhet og manskinlæring med tanke på å knytte samme ideer til et «høyere ordens IDS». Dette kapitlet tar for seg aspekter med konkret valg av hva som skal klassifiseres, og hvilken maskinlæringsmetode som skal nyttes til dette.

Dette betyr at vi vil se på vurderingene som ble gjort rundt valget av å klassifisere falske og reelle alarmer, og valget av «naiv Bayes klassifikator» som maskinlæringsmetode.

4.1 Hva skal klassifiseres

Snort-alarmer må være klassifisert manuelt før de kan benyttes til læring. I vårt case (Telenors system) var disse klassifisert etter alvorlighetsgraden til angrepet, som avgjør hvordan analysepersonell bør håndtere angrepet.

Alternativene våre var derfor å klassifisere falske alarmer og reelle alarmer med alvorlighetsgrader, eller holde oss til kun å skille mellom falske og reelle alarmer.

Siden vi ønsker å vise en generell utforming av et «høyere ordens IDS», vil vi ikke klassifisere alvorlighetsgrad da dette er å knytter oss opp mot vår spesielle case. Vi vil derfor klassifisere Snort-alarmer som enten falske eller reelle alarmer. Dette er også i samsvar med vårt mål utfra hypotesene 1 og 2 i kapittel 1.2. Vi vil benytte begrepet *klasser* som klassifiseringsalternativene til en Snort-alarm. Klassene i vårt tilfelle vil være «Reell Alarm» eller «Falsk Alarm».

I vår case må vi derfor konvertere de opprinnelige klassene (alvorlihetsgradene) til disse klassene før vi kan benytte treningseksemplene i maskinlæringen.

4.2 Valg av klassifiseringsmetode

Av de tre ulike maskinlæringsmetoder vi nevnte i kapittel 3, falt valget vårt på «naiv Bayes klassifikator». Dette er en anerkjent metode innen maskinlæring, og er brukt med stor suksess i andre sammenhenger. Eksempler er tekst-klassifisering og diagnosering. «Popfile» og «Spam Assassin», er to spamfiltreringsapplikasjoner som benytter seg av en utvidet versjon av tekstklassifisering basert på «naiv Bayes».

«Naiv Bayes klassifikator» takler moderate til store mengder data å lære på. En fordel i vår oppgave er at metoden takler mange attributter, siden Snort-alarmer inneholder titalls elementer vi ser som mulige attributter. At noen av attributtene bidrar i liten grad til klassifiseringen, vil ikke være like kritisk som f.eks hos K-nærmeste nabo.

Et annet aspekt ved valg av maskinlæringsmetode er at metodene er såpass like at resultatene mellom disse trolig ikke vil sprike dramatisk. Derfor, når vi implementerer og tester «naiv Bayes klassifikator», vil dette antageligvis gi en generell pekepinn på nyttigheten av å benytte maskinlæringsmetoder til å klassifisere signaturbaserte IDS-alarmer.

«Naiv Bayes klassifikator» er også godt egnet om man ønsker å bygge ut systemet til å lære i sanntid mens det er i bruk. Det vil si at systemet lærer av det analysepersonellet klassifiserer Snort-alarmer til manuelt. K-Nærmeste nabo ville muligens også kunne gjøre dette, fordi læringen bare dreier seg om å legge til treningseksempler i en liste. Et problem er at hele treningseksempelet måtte lagres istedenfor kun en oppdatering i det tallmaterialet som naiv Bayes benytter [1]. Det hadde også krevd en god metode for indeksering. Beslutningstre har ingen mulighet for å læres underveis, da det ville innebære en stor prosess med å forandre trestrukturen.

[1] Tallmaterialet består av opptellinger på hvor mage ganger verdier er sett for hver klasse og antall ganger hver av klassene er sett.

Kapittel 5

Design av treningssett

Med treningssett mener vi her en strukturert samling av treningseksempler som kan nyttes til opplæring av den naive Bayesianske klassifikatoren. Dette kapitlet tar for seg framgangsmåter for å bestemme innholdet i et slikt treningssett slik at det er best mulig egnet til å lære opp klassifikatoren. Vi kaller dette design av treningssett. Denne prosessen vil innebære en nøye vurdering av tilgjengelig treningseksmpler og en kritisk gjennomgang av attributter.

Treningseksemplene i vårt tilfelle er manuelt klassifiserte Snort-alarmer, mens med attributter mener vi de enkelte elementene Snort-alarmene er bygget opp av. Hovedsakelig innebærer attributtene headerfelt fra Internet-protokollen og de mest vanlige transport-protokollene. [1]

Vi kombinerer generelle problemstillinger rundt design av treningssett, med eksempler fra vårt praktiske case med Telenor som har endel mer spesifikke aspekter knyttet til deres bruk av Snort. Dette gir likevel et bilde på hva en kan forvente seg av utfordringer i andre settinger. Eliminering av signaturer og problematikk ved manuelle klassifiseringsrutiner er eksempler på dette.

5.1 Treningssett

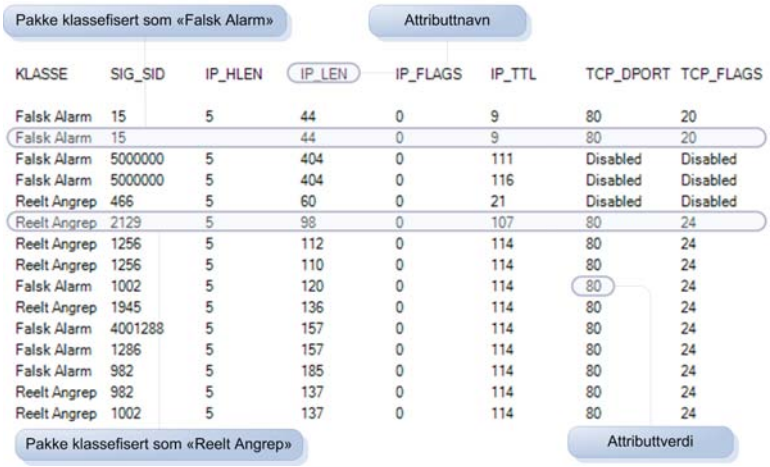
En ferdig strukturert samling med treningseksempler omtales som ett treningssett. Disse dataene må være mulig å hente frem igjen til bruk av klassifikatoren.

Treningssettets innhold er svært viktig da dette er avgjørende for hvor bra klassifikatoren blir til å klassifisere fremtidige Snort-alarmer (i vårt tilfelle). Det er derfor viktig å gjøre en nøye vurdering av tilgjengelig attributter, for å finne hvilke som bør benyttes og hvilke som bør unnlates fra

[1] I vårt tilfelle har vi TCP, UDP og ICMP-protokollene. Det er en forenkling å kalle ICMP for transportprotokoll, men den ligger direkte over IP rent arkitektur-messig, og således av samme natur som UDP og TCP.

treningsettet. Treningssettet bør representere et realistisk utsnitt av virkeligheten, slik at klassifikatoren er godt rustet til å klassifisere nye Snort-alarmer basert på dette.

Dårlige treningssett kan bære preg av mye støy, attributter som forvirrer, eller treningssettet som er lite variert (eller for små). Treningssett som er små eller lite varierte vil representere en liten del av virkeligheten, og vil derfor føre til at klassifikatoren kan møte på Snort-alarmer senere som den ikke har grunnlag for å klassifisere.



Figur 5.1: Eksempel på treningssett

Figur 5.1 viser et utdrag av vårt treningssett. Dette er en enkel tekstfil der treningseksemplene, i form av klassifiserte Snort-alarmer er strukturert linje for linje.

5.2 Eliminering av identiske alarmer

I vårt case utløser enkelte angrep vesentlig flere Snort-alarmer enn andre. Det kan variere fra å utløse over tusen til kun en enkelt Snort-alarm for ett angrep. Her vil vi kalle slike rekker av Snort-alarmer for «identiske Snort-alarmer». Siden disse stammer fra samme angrep, vil klassen («Reell alarm» eller «Falsk alarm») være lik for alle disse. Benyttes alle slike «identiske Snort-alarmer» ukritisk i treningssettet, vil dette føre til at angrep som utløser mange alarmer får en uforholdsmessig stor innflytelse på læringsprosessen.

Dette kan igjen føre til problemer som at enkelte attributtverdier vil kunne gi en sterk antydning til klasse, på et feilaktig grunnlag. Dette fordi enkelte attributter gjerne vil ha lik attributtverdi for alle Snort-alarmer som stammer

fra samme angrep (som f.eks portnummer). Om denne attributtverdien i realiteten ikke har noe med klassen å gjøre, men opptrer tilfeldig, vil likevel klassifikatoren tro annerledes siden verdien nå er sett tusenvis av ganger for klassen alarmene i dette angrepet. Det er derfor en fordel å begrense «identiske Snort-alarmer» i treningssettet.

Vårt system er implementert slik at det kun blir godtatt fem «identiske Snort-alarmer» i treningssettet. Skulle det opprinnelig være flere enn dette, blir disse fjernet. «Identiske Snort-alarmer» kjente vi igjen ved at de er påfølgende med hensyn på tid, og har samme signatur samt kilde- og destinasjonsIPer.

5.3 Vurdering av attributter

Attributter er de forskjellige elementene som inngår i selve Snort-alarmen som vi skal klassifisere. I dette delkapitlet diskuterer vi hvilke hensyn en bør ta ved valg av attributter og hvilke vi valgte til vårt praktiske case. I lignende verktøy til Snort, vil man ende opp med stort sett samme mulige attributter, da hovedvekten er header-felt i kommunikasjonsprotokoller. Våre antagelser er da gjeldene generelt og ikke utelukkende i et system som baserer seg på Snort.

Vi kjenner til naiv Bayes klassifikator sin styrke framfor beslektede algoritmer at den tåler til en viss grad støy, og kan benytte mange attributter til trening. Likevel bør attributter vi velger å ta med i treningssettet vurderes nøye, slik at man ikke skaper unødig støy i treningssettet. Mange intetsigende attributter vil også gjøre både læringsprosessen og klassifiseringen mer ressurskrevende enn nødvendig. Med intetsigende mener vi attributter som i liten eller ingen grad påvirker klassifiseringen. I denne seksjonen vil vi derfor gi en grundig vurdering av attributtvalg.

De enkelte elementer en Snort-alarm består av, kan benyttes som attributter i treningssettet. Dette inkluderer felt fra IP-headeren, og transportprotokollen (TCP/UDP/ICMP) samt informasjon fra Snort og payloaden som følger med IP datagrammet. Headerfelt og Snort-informasjon innbefatter tilsammen 31 ulike kandidater som kan benyttes. I tillegg kan payloaden deles opp i ett ønskelig antall attributter. Figur 5.2 viser en liste over mulige attributter fra Snort og headerfelt i transportprotokollene.

5.3.1 Unike attributtverdier

Unike attributter vil si attributter med verdier som ikke kan forekomme ved senere anledning. Unike attributter bør unngås i treningssettet fordi disse aldri kan påvirke klassifiseringen i noen retning. Isteden vil de forårsake

Alarm nummer	Sensor id
Tidspunkt	Signatur id
Signatur revisjon	Signatur navn
Signatur priotitet	IP source
IP destination	IP header length
IP type of service	IP lenght
IP flags	IP time to live
IP protokoll	TCP source port
TCP destination port	TCP reserved
TCP flags	TCP window
TCP urgent	TCP Ack number
TCP lenght	TCP options
TCP sequence number	UDP source port
UDP destiantion port	UDP lenght
ICMP type	ICMP code

Figur 5.2: Mulige attributter fra Snort i vårt case.

mer bruk av systemressurser ved opplæring av klassifikatoren. Attributtene med tilnærmet unike attributtverdier kan i sjeldne tilfeller påvirke resultatet, men da i mer eller mindre tilfeldig retning. Vi velger derfor å fjerne slike tilfeller også. I vårt case fjernet vi følgende attributter som følge av denne argumentasjonen:

- Alarm-nummer. I våre treningsempler var alle Snort-alarmer merket med et løpende nummer siden dataene var lagret i en database. Dvs en unik id. Slike elementer må fjernes fra treningssettet.
- Tidspunkt, attributten er unik da den inneholder dato og klokkeslett inkludert årstall.
- TCP/UDP source port. Siden avsenderporter ofte er tilfeldige, vil disse ikke bidra til riktig klassifisering mellom «Falsk alarm» eller «Reell Alarm».
- IP source. Å trene klassifikatoren til å tro bestemte IP-adresser er «farlige» kan bli feilaktig, da slik info er svært flyktig i dagens Internet. F.eks får et stort antall av datamaskinene i verden dynamisk utlevert ip-adresser. Siden det finnes enormt mange IP-adresser i forhold til hva som vil være med i ett treningsett vil også verdien kunne fortone seg som tilnærmet unik
- TCP sequence/ack nr. Sekvensnummer er et tilfeldig bestemt tall som blir satt ved opprettelsen av en TCP forbindelse. Dette vil derfor være

unikt. TCP ack vil også være unikt, da dette er en retur av innkommende sekvensnummer for å bekrefte at et segmentet er mottatt.

5.3.2 Testing av attributter

De resterende attributter har vi vurdert enkeltvis. Vi har benyttet testing av de enkelte attributter, for å avgjøre deres nytteverdi. Testen består i å lage treningssett der vi kun har med denne attributten vi vil teste samt klassen (fasiten). Vi trener en klassifikator på dette minimum av informasjon for å teste hvor god de enkelte attributtene er til å klassifisere Snort-alarmer alene. Testmetoden her er kryssvalidering som det står mer om i kapittel 6.5.

Attributter som gir forholdsvis dårlige resultater ved testing, vil vi vurdere hvorvidt bør fjernes fra treningsettet. Er disse av en slik natur at de skaper unødig støy, eller alltid vil gi lite bidrag til klassifiseringen, fjernes attributten. Ellers ønsker vi å benytte flest mulig av attributtene i treningsettet. Drøft av hver enkelt attributt er såpass case-avnhengig at vi velger å ikke ha dette med i selve rapporten. Vedlegg C tar for seg dette.

Hvilke attributter vi endte opp med å bruke summerer vi opp i kapittel 5.3.4.

5.3.3 Payload

Payloaden er datamengden som sendes over ved et transportlags-segment. Siden Snort-alarmer ofte vil ha bestemte mønstre av payload som indikerer «Falsk Alarm» eller «Reell Alarm», har vi valgt å benytte payload i treningssettet i tillegg til nevnte attributter.

Payloaden er spesiell fordi den kan inneholde svært mye data. Det er derfor nødvendig å begrense denne mengden. En attributt med payload tildeles derfor en liten del av den totale mengden av payloaden. For stor del payload som attributtverdier, vil gjøre det mindre sannsynlig at attributtvediene går igjen i flere av Snort-alarmene. Dette vil igjen føre til at dette ikke vil bidra med nyttig informasjon ved klassifisering.

For likevel å få med en viss mengde payload, kan vi dele opp en større mengde payload på flere attributter. Vi gjorde tester der vi varierte payload mengde og antall attributter denne ble fordelt utover for å komme fram til et godt resultat som kunne benyttes i treningssettet.

Ut fra resultater av denne testingen, endte opp med å benytte fire attributter til payload. Hver av disse fikk åtte bytes av payloaden, tatt fra begynnelsen av payloadfeltet. Selv om dette ga oss gode resultater, er det mange mulige kombinasjoner her, og flere av våre tester ga svært like resultater. Vi fremmer derfor ingen påstand om at dette er optimalt med hensyn på bruk av payload som attributter.

5.3.4 Oppsummering

Fra attributt-testing og diskusjon rundt bruk av payload og unikhet, endte vi opp med å benytte disse attributtene:

Signatur id	IP destination
IP header lenght	IP lenght
IP flags	IP time to live
TCP destination port	TCP flags
TCP window	TCP urgent
UDP destiantion port	UDP lenght
ICMP code	ICMP type
+4 felt med payload	

5.4 Direkte avhengighet mellom attributter

Om klassen til en Snort-alarm ikke er gitt vil det være en logisk avhengighet mellom attributtverdiene i denne Snort-alarmen. For å illustrere dette kan en tenke seg at alle attributtverdier i en Snort-alarm, med unntak av en, har verdier som peker i retning av klassen «a». Da vil sannsynligheten for at siste attributt også har en verdi som peker i retning av klasse «a» være større enn for alle andre klasser. Tenker vi oss at klassen er gitt isteden, vil vi kunne si at denne avhengigheten er mellom klassen og attributtverdiene istedenfor mellom attributtene. Denne avhengigheten er noe vi ønsker, da dette vil være nødvendig for at metoden skal fungere.

Tilfellet der det forekommer en direkte avhengighet mellom to eller flere attributter uavhengig om klassen er gitt eller ikke, er derimot noe vi helst vil unngå i treningssettet. Det vi si tilfeller der en verdi for en attributt entydig bestemmer, eller i stor grad påvirker, en annen.

Den naive Bayesianske klassifikatoren gjør en tilnærming ved å anta at det ikke forekommer noen slik direkte avhengighet mellom attributtene i treningssettet. Dette vil i mange tilfeller ikke stemme, da flere klare og mindre klare sammenhenger vil forekome. Alle attributter som kan sies å bli påvirket av felles kilde vil egentlig ha små direkte avhengigheter. Eksempel på dette kan være et hacker-verktøy som setter fast «IP time to live» og TCP port vil skape en liten avhengighet mellom disse feltene.

I treningssettet vårt basert på attributtvalg som nevnt i kapittel 5.3, oppstod det klare direkte avhengigheter mellom attributter som har TCP, UDP og ICMP data og mellom payload attributtene.

TCP, UDP og ICMP attributter vil konkret si[2]:

[2]TCP porter og UDP porter trenger ikke være adskilte attributter, men er å anbefale da dette blir mer spesifikt ved læring av klassifikator.

TCP source port	TCP destinatiotion port
TCP reserved	TCP flags
TCP window	TCP urgent
TCP lenght	UDP lenght
UDP source port	UDP destination port
ICMP code	ICMP type

Om det i en Snort-alarm er TCP som transportprotokoll betyr det at UDP og ICMP attributtvediene vil stå tomme. Tomme attributtverdier vil også trenes på om vi om vi ikke aktivt gjør noe med dette. Dermed vil flere attributter gi samme informasjon, slik at dette vektes for mye når vi trener klassifikatoren.

På samme måte vil payload-attributtene kunne være avhengig av hverandre om ikke alle disse har en verdi. F.eks vil et IP-datagram uten payload bety at alle payload-attributtene står tomme. Ved at første payloadattrbutt er tom vil bety at de tre andre også må være tomme.

Siden disse direkte avhengighetene som nevnt her er såpass klare er dette noe vi ønsker å gjøre noe med. Ved å merke de attributtverdier som skapes av direkte avhengighet i treningssettet med «Disabled» kan læringsprosessen implementeres slik at disse ikke blir tatt hensyn til når vi trener klassifikatoren. Det vil si at klassifikatoren lar være å telle disse.

5.5 Case-relaterte utfordringer

5.5.1 Automatisk klassifisering av Snort-alarmer

I noen tilfeller vil det være ønskelig å behandle spesielle Snort-alarmer automatisk, uten de går til manuell analyse. Dette fordi enkelte signaturer alltid blir behandlet likt. I vårt case med Telenor fantes det en modul som gjorde slik automatisk klassifisering av enkelte Snort-alamer (se forøvrig kapittel 1.4). Snort-alarmer som fjernes er gjerne basert på signatur. Siden disse Snort-alarmene allerede blir riktig klassifisert, behøver ikke disse å bli behandlet av klassifikatoren. Av denne grunn bør det heller ikke trenes på disse Snort-alarmene, fordi treningseksempler skal være eksempler på oppgaven til klassifikatoren. Derfor fjerner vi alle Snort-alarmer som blir behandlet automatisk fra treningsettet. I vår case var dette en enkel sak da Snort-alarmer som ble automatisk behandlet var merket spesiellt i den databasen der vi hentet våre treningseksempler.

5.5.2 Eliminering av signaturer

I vårt case måtte vi gjøre noen grep da enkelte signaturer kunne være overrepresentert som feilkilder. Det var flere årsaker til dette, men hovedgrunnen var mangel på konsistens i den manuelle klassifiseringen. Dette

skyldes uklar policy på klassifisering av Snort-alarmer med spesifikke signaturer eller menneskelige feil.

Et godt eksempel er Snort-alarmer som under perioden disse er hentet fra har blitt nedrangert som sikkerhetsrisiko, og derfor blitt klassifisert ulikt under denne perioden. F.eks vil en Snort-alarm som ble holdt som «farlig» i uke «a» bli logget som angrep. Senere, i uke «b», kan mer informasjon rundt Snort-alarmen føre til at den nå regnes som ufarlig, og derfor bli klassifisert som falsk Snort-alarm.

Vi kjørte egne tester der vi analyserte de Snort-alarmene som ble feilklassifisert med hensyn på signaturer. De største «problemsignaturene» fikk vi dermed fjernet fra treningssettet. Ved å fjerne signaturer fra treningsettet på denne måten, kan det se ut til at vi ser bort ifra et problem med støyende signaturer. Resultatet ved dette er at Snort-alarmer med disse signaturene ikke vil bli trent på. Dermed kan dette gå utover klassifiseringen av disse. Problemet kan løses hvis policy på klassifiseringen utbedres, slik at et nytt treningsett uten denne feilen kan lages.

5.5.3 Klassifiseringsrutiner

Et angrep vil ofte bestå av en serie med ip-datagram som alle trigger samme signatur. Slike angrep forårsaker også da gjerne en rekke like Snort-alarmer. Analysepersonell vil kunne se på dette som ett enkelt angrep og derfor nøye seg med å klassifisere en av disse Snort-alarmene som angrep.

I vårt case ved Telenors sikkerhetstjeneste, blir Snort-alarmer klassifisert på denne måten. Her er det en fordel, da dette gjør at Snort-alarmer som er klassifisert i deres database kan rapporteres til kundene, uten å risikere at det samme angrepet blir rapportert flere ganger. Situasjonen er derfor at en stor mengde av «Reell alarm», og alle av «Falsk alarm» forblir uklassifisert. Dette betyr at det meste av de tilgjengelige Snort-alarmene faktisk er «feil» klassifisert om man ser på hver Snort-alarm enkeltvis.

For å kunne benytte Snort-alarmer som treningseksempler, var vi tvunget til å rette det som var feilklassifisert manuelt. Dette er en tidkrevende prosess som i stor grad satt en begrensning størrelsen av treningssettet vi kunne jobbe med. Vi regner også med at dette førte til ekstra støy da det er lett å gjøre feil i slike prosesser.

5.6 Mulige feilkilder i treningssettet

Ved utarbeidelsen av treningssett dukket det opp flere momenter som vil påvirke treningssettet i negativ retning. Dette er problemer som ofte vil forekomme ved manuell klassifisering av Snort-alarmer, som hos oss var en nødvendighet for å lage treningssett. Forslag til muligheter for å forbedre dette kommer vi tilbake til i kapittel 9. Feilkildene som nevnes i denne

seksjonen vil bli tatt med i betraktingen når vi drøfter resultatene av hvor god klassifikatoren er til å klassifisere Snort-alarmer i kapittel 7.

5.6.1 Lite variasjon i treningssettet

Variasjon i treningsettet er viktig for å forberede klassifikatoren best mulig til den store variasjonen av Snort-alarmer som finnes. Et variert treningsett presenterer et større spekter av attributtverdier, som er kjernen i hva klassifikatoren benytter til å avgjøre klasse.

Mengden av Snort-alarmer tilgjengelig i vårt tilfelle begrenset seg naturlig som følge av at Snort-alarmer ikke var fulstendig klassifisert, slik at vi måtte foreta en tidkrevende manuell klassifisering. Begrenset mengde med treningseksempler begrenser også muligheten for et variert treningssett.

Metoder for å variere treningssettet kan være å benytte en viss minimumsmengde av Snort-alarmer for både klassen «Reell Alarm» og «Falsk Alarm» for hver signatur. Dette sørger for at treningsettet i større grad vil dekke ulike aspekter ved de forskjellige agrepsformene. Å benytte Snort-alarmer fra et bredere tidsspekter er en annen metode som vil kunne gjøre variasjonen i treningsettet større. Dette er et mulig forbedringspotensiale som er uprøvd fra vår side.

5.6.2 Støy i treningsettet

Vårt treningssett vil inneholde en del støy i form av feil klassifiserte Snort-alarmer. Dette skyldes elementer vi tidligere har nevnt som varierende policy for klassifisering av Snort-alarmer, og den manuelle behandlingen vi måtte gjøre i treningssettet av case-relaterte årsaker.

En annen feilkilde ved vår case er at da flere nettverk blir monitorert samtidig, vil en Snort-alarm kunne klassifiseres ulikt ettersom på hvilket nettverk den inntraff. Dette kommer av detaljert kjennskap til nettverkene. F.eks. vil en alarm på et web-angrep som vanligvis klassifiseres som «Reell Alarm» kunne avblåses som «Falsk Alarm» fordi det ikke eksisterer noen webserver i dette nettverket etc.

Et siste poeng er at manuell klassifisering alltid vil kunne medføre noe feil av menneskelig karakter. Men å takle enkeltfeil blant mengden treningseksempler, er en av styrkene til vår metode og et av de beste motivene for valg av naiv bayesiansk klassifikator.

Kapittel 6

Implementasjon av «høyere ordens IDS»

Dette kapitlet tar for seg funksjonaliteter som er implementert i vårt «høyere ordens IDS». Disse kan betraktes som enkeltstående funksjoner som sammen vil kunne trene opp en naiv Bayes klassifikator utfra tilgjengelige treningseksempler (Snort-alarmer). Vi går også gjennom funksjoner for å teste hvor mye klassifikatoren vil klassifisere riktig med hensyn på «Falsk Alarm» og «Reell Alarm».

Vi har valgt å fokusere på argumentasjon for metodene våre framfor å detaljert beskrive programmeringstekniske løsninger spesifikke for dette tilfellet. Funksjonene i vårt verktøy vi skal beskrive:

- *Konvertering av treningseksempler* – Lager treningssett utfra gitte treningseksempler.
- *Læringsfunksjonen* – Benytter treningssettet til læring.
- *Klassifikatoren* – Benytter resultatet fra læringsfunksjonen til å klassifisere alarmer.
- *Underklassifisering* – funksjon for å utvide antall klasser for mer presis klassifisering.
- *Testfunksjoner* – metoder som benyttes til å gi målbare resultater av kvaliteten på klassifikatoren. Det vil si hvor mye den kan klassifisere riktig av gitte Snort-alarmer med hensyn på klassene «Falsk Alarm» og «Reell Alarm».

6.1 Konvertering av treningseksempler

Våre treningseksempler er resultat av rå eksport av loggede Snort-alarmer fra Telenors databasesystemer, lagret i en tekstfil. Treningseksemplene

trenger behandling før de kan inngå i et treningssett, siden treningssettet som kjent skal benyttes til å trene en naiv Bayes klassifikator.

Andre scenarier er at slik data ligger lagret i systemfiler, forskjellige database-systemer, i tekstfiler eller lignende. En form for konvertering av treningseksemplene vil i alle tilfeller være nødvendig, fordi ikke all informasjon her vil være ønskelig å benytte til trening.

Hovedoppgaven til funksjonen for konvertering av treningseksempler er:

- Fjerne attributter.
- Henter ut ønsket mengde payload.
- Fjerne støyende signaturer.
- Begrense «identiske alarmer».
- Markere attributter med direkte avhengighet.
- Formatere treningseksempler så det passer malen satt for treningssettet
- Lagrer treningssettet i ønsket format

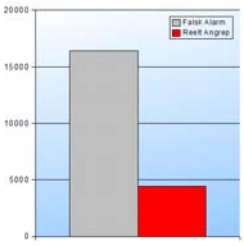
Ved å benytte denne funksjonen har vi nå klart et treningssett som kan benyttes av læringsfunksjonen.

Forklaring rundt konverterings-funksjonens oppgaver er beskrevet i større detalj i kapittel 5.

6.2 Læringsfunksjonen

Læringsfunksjonen leser treningssettet og henter de statistiske data klassifikatoren trenger for å klassifisere Snort-alarmer. Dette dreier seg om en opptelling av hvor mange ganger hver verdi for hver attributt finnes for hver klasse i treningssettet.

Eksempel. *I vårt treningssett kan vi tenke oss at attributt «IP destination» har en attributtverdi «111.222.333.444», som er sett 30 ganger for klassen «Falsk Alarm» og 80 gang for klassen «Reell Alarm». Dette er informasjonen læringsfunksjonen må telle opp og huske.*



Læringsfunksjonen må også telle opp hvor mange instanser det er av hver klasse i treningssettet. For igjen å bruke vårt tilfelle som eksempel, hadde treningssettet 16405 av Snort-alarmer med klassen «Falsk Alarm» og 4461 for «Reell Alarm».

Etter at læringsfunksjonen har lest hele treningssettet, lagres dette på en strukturert måte i

minne tilgjengelig for senere bruk av klassifikatoren. Vi kaller gjerne prosessen til læringsfunksjonen for å «trene opp en klassifikator».

6.2.1 Hensyn til direkte avhengighet mellom attributter

Som vi beskrev i kapittel 5.4 vil enkelte attributtverdier ikke være ønskelig å trenes på av hensyn til direkte avhengighet mellom attributter. Disse setter vi til «disabled» i treningssettet som læringsfunksjonen skal kunne kjenne igjen. Når slike verdier oppstår skal disse ikke telles. Vi har dermed løst problemet med direkte avhengighet mellom attributter i treningssettet.

6.2.2 Lagring av data fra læringsfunksjonen

Det kan være hensiktsmessig å implementere en metode for å lagre resultatet fra læringsfunksjonen i en database. Dermed unngår vi at læringsfunksjonen må benyttes på nytt hver gang systemet startes opp.

Vi har valgt å implementere dette ved å bruke en SQLite [11] database til lagringen. Lagringsmodulen er utviklet slik at den lett kan til å bruke andre SQL baserte databaser.

Senere i rapporten diskuterer vi en mulighet for å lære i sanntid (kapittel 9.2.1). Her vil det være påkrevd med en mulighet for en lagringsfunksjon.

6.3 Klassifikatoren

Klassifikatoren er den funksjonen som klassifiserer Snort-alarmer utfra det statistiske materialet læringsfunksjonen har hentet fra treningssettet. Denne funksjonen baseres på teori presentert i kapittel 3.4.2(side 24), men vi vil her forklare virkemåten til klassifikatoren i større detalj.

Når en alarm skal klassifiseres, ser klassifikatoren på attributtverdiene i alarmen. Fra læringsfunksjonen får vi antall ganger disse attributtverdiene, a_i, opptrer for hver klasse, k_j, som kan skrives $|(a_i|k_j)|$. Læringsfunksjonen gir også antall forekomster av hver klasse $|k_j|$ i treningsettet. Dermed kan vi regne ut sannsynligheten til en attributtverdi gitt klasse, på følgende måte:

$$P(a_i|k_j) = \frac{|(a_i|k_j)|}{|k_j|}$$

Dette kan sees på som sannsynlighetsbidraget denne attributtverdien gir for at gjeldene klasse er korrekt. Ved å regne ut dette for alle attributtverdier i Snort-alarmen for alle klasser, kan klassifikatoren benytte dette i formelen:

$$k_{NB} = \underset{\in K}{\operatorname{argmax}} \prod_i P(a_i|k_j)$$

Klassen der produktet av sannsynlighetsbidragene fra hver attributtverdi er høyest, representerer resultatet av klassifiseringen.

Sammenligner vi formelen over med formelen 3.10 i kapittel 3 ser vi at vi her har valgt en tilnærming der vi setter $P(k_j) = 1$. Dette er vanlig å gjøre ved bruk av naiv Baye i sammenhenger der sannsynligheten for klassen er vanskelig å fastslå, noe som vi mener er generellt for signaturbaserte IDS[1].

I våre treningseksempler varierer forholdet mellom falske Snort-alarmer og reelle fra dag til dag.

6.3.1 Minsteverdi for en attributtverdi

Som nevnt i kapittel 3.4.2 må vi ta spesielt hensyn til tilfeller der en av attributtverdiene a_i finnes for en av klassene k_j i treningssettet, men ikke den andre. Vi kaller denne spesielle attributtverdien $\hat{a_i}$ og klassen denne ikke eksisterer for $\hat{k_j}$.

Siden attributtverdien $\hat{a_i}$ ikke finnes for klassen $\hat{k_j}$ i treningssettet, resulterer dette i $P(\hat{a_i}|\hat{k_j}) = 0$. Da sannsynligheten for hele klassen $\hat{k_j}$ bestemmes ved å multiplisere $P(a_i|\hat{k_j})$ for alle a_i, vil dette resultere i sannsynlighet lik null for klassen $\hat{k_j}$.

Tenker vi oss at alle attributtverdier utenom $\hat{a_i}$ gir god sannsynlighet for at riktig klasse er $\hat{k_j}$, vil likevel $\hat{a_i}$ overstyre dette ved å nulle ut den totale sannsynligheten. Som følge av dette vil vi aldri kunne klassifisere en Snort-alarm med verdi $\hat{a_i}$ til klasse $\hat{k_j}$[2].

Minsteverdi vil altså si å sette den laveste tillatte verdien for $P(\hat{a_i}|\hat{k_j})$, slik at denne ikke vil være null i slike tilfeller.

Når bør minsteverdi benyttes

Et treningssett som gir en perfekt gjenspeiling av virkeligheten, vil ikke kreve en klassifikator med en definert minsteverdi. En attributtverdi som ikke er sett for en klasse ville i dette tilfellet bety at attributtverdien heller ikke kan finnes for denne klassen, slik at sannsynligheten for denne klassen skal bli null.

Treningssett som inneholder komplekse treningseksempler som Snort-alarmer vil derimot alltid ha mangler i forhold til virkeligheten. Dette skyldes

[1] Denne påstanden baserer vi på generelle observasjoner og erfaring gjort hos Telenor Sikkerhetssenter.

[2] Om da ikke de øvrige klassene også har en attributt som gir sannsynlighetsbidrag lik null.

elementer som lite variasjon i treningssettet, sjeldne attributtverdier og tilfeldigheter. Dette vil derfor være hovedårsakene til at attributtverdier ikke er representert ved alle klasser.

Derfor er det viktig å bestemme en minsteverdi for alle tilfeller der treningssettet kan sies å være et mindre utsnitt av virkeligheten.

Bestemme størrelsen på minsteverdien

Treningssett vil ofte ha ulik fordeling av klasser. I vårt tilfelle ser vi dette med at det er 16404 av klassen «Falsk Alarm» og 4461 av «Reell Alarm». Dette betyr at klassen «Falsk Alarm» vil ha et størst spekter av attributtverdier enn «Reell Alarm». Med andre ord må klassifikatoren oftere benytte minsteverdi når den beregner sannsynligheten for kassen som har færrest forekomster i treningsettet.

Når vi har utført tester på ulike minsteverdier har vi derfor lagt spesielt vekt på riktig klassifisering av klassen «Reell Alarm» da denne oftest blir påvirket av minsteverdien.

I vedlegg A.1 dokumenterer vi tester på minsteverdien, der denne er satt statisk til en fast verdi i klassifikatoren. Vi gjentok denne testen med ulike minsteverdier. Ved lave minsteverdier ser vi klassen «Reell Alarm» bli klassifisert dårligere. Grunnen til det er at da vil minsteverdien redusere sannsynligheten til den klassen der den oppstår. Dette vil oftest gjelde «Reell Alarm». Ved høye minsteverdier ser vi klassifiseringen blir svært dårlig totalt sett. Likevel får vi her motsatt effekt slik at klassen «Falsk Alarm» påvirkes mest.

Vedlegg A.2 viser en test der vi lot klassifikatoren regne ut minsteverdien basert på antall forekomster av hver klasse i treningssettet. Dette gjorde vi slik at ved tilfeller der en attributtverdi ikke finnes for en klasse, blir denne behandlet som om den finnes en enkelt gang for klassen. Vi får uttrykket:

$$m_k = \frac{1}{|k|} \tag{6.1}$$

Der m_k er minsteverdien til klassen k og $|k|$ = Størrelsen av klassen

Dette gav et bedre resultat for riktig klassifisering av klassen «Reell Alarm» i forhold til forrige test.

Grunnen til at denne metoden for å bestemme minsteverdi fungerer best, er at denne dynamisk tilpasser minsteverdien etter antall forekomster av klassene i treningssettet. Å la minsteverdien tilsvare en attributtverdi som opptrer en enkelt gang for gjeldene klasse i treningssettet er en tilnærming som viser seg å fungere bra.

Med tanke på at systemets nyttighetsverdi er avhengig av å i stor grad klassifisere reelle alarmer korrekt, falt valget på sistnevnte metode. Dette

har også sin klare fordel av at man slipper å bestemme minsteverdien spesifikt for hvert nytt treningssett som måtte bli designet, siden klassifikatoren regner denne ut selv.

6.3.2 Sikkerhetsmargin

Som vi har sett blir klassifiseringen til den naive Bayesianske klassifikatoren basert på utregnede sannsynligheter for klassene. I utgangspunktet vil metoden ukritisk returnere den klassen med høyst sannsynlighet. Ved å se på forholdet mellom sannsynlighetene for de ulike klassene ved en klassifisering av en Snort-alarm, kan vi få et mål på hvor sikker klassifikatoren er i dette tilfelle.

Antar vi at en lav sikkerhet ved klassifisering betyr større sannsynlighet for feil klassifisering, vil hypotese 3 i innledningskapittelet motivere oss til å ta spesielt hensyn til dette ved klassen «Reell Alarm». Vi rekapitulerer hypotese 3:

Hypotese 3. *Feilklassifiserte reelle alarmer vil kunne reduseres kraftig ved ulike metoder innenfor maskinlæring og fortsatt kunne ha en akseptabel effekt til å stoppe falske alarmer.*

Det vi ønsker er å kunne kontrollere klassifiseringen ved lav sikkerhet, slik at disse Snort-alarmene ikke blir klassifisert som «Reell Alarm».

Ved å la klassifikatoren beregne grad av sikkerhet ved hver klassifisering, kan vi innføre en terskel som sier hvilken margin som kreves for å kunne klassifisere en alarm. Denne terskelen kaller vi for sikkerhetsmargin.

Om et forsøk på klassifisering mislykkes som følge av sikkerhetsmarginen, resulterer klassifiseringen i «usikker». Dermed kan vi behandle disse usikre klassifiseringene på lik linje som reelle alarmer, ved å sende disse alarmene til manuell analyse. Grunnen til å klassifisere disse som «usikker» istedenfor «Reell Alarm» direkte er av hensyn til testing, da vi gjerne vil se omfanget av usikre resultater ved ulike sikkerhetsmarginer.

Naiv Bayes klassifikator har en karakteristikk ved at den kan få svært ulike sannsynligheter for de ulike klassene [3]. Ved å ta forholdet mellom disse vil også resultere i en høy sikkerhet. Derfor må sikkerhetsmarginen, som kan presenterers i prosent, settes urealistisk høyt. Beste metode for å bestemme denne er derfor ved testing.

Når sikkerhetsmarginen økes, vil vi minske sjansen for å feil klassifisere reelle alarmer, som er ønsket effekt. Bivirkningen er at endel «usikre» klassifiseringer som skulle vært «Falsk Alarm», nå blir behandlet som en «Reell Alarm». Dette betyr at færre av klasen «Falsk Alarm» oppdages når sikkerhetsmarginen øker.

[3] Dette skyldes at naiv Bayes klassifikator ikke tar hensyn til tilfeller der flere attributter påvirkes av en felles kilde. Det vil si små avhengigheter mellom attributter blir skjult for klassifikatoren

6.4 Underklassifisering

Underklassifisering er en teknikk som benyttes på treningssettet for å dele hver klasse i flere underklasser. Alle Snort-alarmer i treningssettet som får samme underklasse har likhetstrekk som gjør fordelingen naturlig. Hensikten med denne teknikken er at hver underklasse fremhever karakteristikker innenfor den opprinnelige klassen. Dette kan hjelpe den naive Bayesiske klassifikatoren til å klassifisere Snort-alarmer korrekt. Vi skal nå forklare fremgangsmåten for å innføre underklasser i et treningssett.

Prosessen starter med å dele det opprinnelige treningsettet opp i et nytt treningsett for hver klasse. Vi får altså to treningsett: et for falske alarmer og et for reelle alarmer. Disse nye treningsettene har dermed felles klasse i utgangspunktet. Klassene blir nå erstattet med helt nye klaser som tildeles tilfeldig. Ønsker vi å dele hver klasse i to underklasser kan et eksempel på nye klasser være «Falsk Alarm_1» og «Falsk Alarm_2». Hver Snort-alarm i treningsettet for falske alarmer vil få klassen erstattet med en av disse to basert på tilfeldig valg.

Ved læringsfunksjonen som beskrevet tidligere i dette kapitlet, vil en klassifikator trenes opp på det nye treningssettet. Denne klassifikatoren benyttes deretter til å klassifisere hver Snort-alarm i det samme treningssettet. Blir Snort-alarmen klassifisert riktig gjøres ingenting, ved feil vil klassen til Snort-alarmen blir forandret i treningssettet til det klassifikatoren gav som resultat.

Når hele settet er gjennomgått på denne måten, trenes klassifikatoren opp på nytt på det nå modifiserte treningssettet og prosessen gjentas. For hver gang man gjentar dette vil det oppstå færre feilklassifiseringer, inntil alle Snort-alarmer klassifiseres korrekt. Da ansees underklassifiseringen for å være endelig. Treningssettene med ferdig underklassifisering, blir tilslutt satt sammen til ett felles sett igjen. Vi står igjen med et treningssett med flere spesialtilpassede klasser som fremhever ulike sammenhenger i de opprinnelige klassene. Antall underklasser som benyttes kaller vi for nivåer av underklasser. Hvor mange nivåer som er optimalt kan bare avgjøres ved testing.

Da underklassene blir satt tilfeldig i utgangspunktet, vil endelige underklassifiserte treningssettet ende opp forskjellig for hvert forsøk. Derfor sier vi det finnes flere forekomster av underklassifiserte treningssett for hvert nivå. Noen forekomster av underklassifisert treningssett vil være bedre enn andre.

For å finne en god forekomst av mulige underklassifisete treningssett, implementerte vi en funksjon som automatisk letet etter dette. Den fungerer slik at den underklassifiserer et treningssett og tester hvor god en klassifikator trent på dette treningsettet er. Funksjonen forsøker med ulike nivåer av underklasser og lager flere forekomster av underklassifiserte treningsett for hvert nivå. Den beste forekomsten av underklassifiserte treningssettet

blir tatt vare på ved å lagres til fil.

6.5 Testfunksjoner

En naiv Bayes klassifikator vil være spesielt dyktig til å klassifisere sine egne treningseksempler. En test på riktig klassifisering av de Snort-alarmene som ligger i treningssettet vil derfor ikke gi noe godt mål på hvor god klassifikatoren er generelt.

Vil vi beskrive to ulike metoder som kan benyttes for å gi et mål på hvor god klassifikatoren er.

6.5.1 Kryssvalidering

Kryssvalidering utføres ved å dele treningssettet i n like deler. En av disse n delene benyttes som testsett, og de resterende $n-1$ delene benyttes som treningssett. Alle Snort-alarmene i testsettet blir så klassifisert. Ved å sjekke klassifiseringen mot klassen (fasiten) vil en se om klassifiseringen var riktig. Dette gjentas n ganger, til alle delene er brukt som testsett en gang. Resultatet av denne testen får vi ved å summere delresultatene fra de n testene.

Hvor mange riktige klassifiseringer som ble gjort totalt vil nå gi en god indikasjon på hvor god klassifikatoren er generelt, fordi vi hele tiden har adskilt Snort-alamene som benyttes til testing og trening.

Likevel må man være observant på feilkilder. En slik er om treningssettet inneholder mange like instanser. Dette vil føre til en likhet mellom treningssettet og testsettene, som igjen gir feilaktig gode testresultater.

Dette er en svakhet ved kryssvalidering som gjør det hensiktsmessig med en alternativ testmetode som kan verifisere resultatene fra kryssvalideringen. Ved å fjerne lange rekker med identiske Snort-alarmer slik vi beskrev i kapittel 5, vil vi dog redusere problemet med likhet mellom testsett og treningsett ved kryssvalidering.

6.5.2 Uavhengig testsett

En annen måte å teste klassifikatoren på, er å ha et eget uavhengig testsett som inneholder Snort-alarmer som kun benyttes til testing. Det vil si disse ikke vil bli trent på overhodet. Snort-alarmene i testsettet vårt ble hentet fra en annen tidsperiode en de i treningssettet. Dette kan avdekke om klassifikatoren er i stand til å klassifisere Snort-alarmer som forekommer utenfor det tidsrommet treningssettet dekker. Dette er ikke gitt på forhånd da nettverkstrafikk og angrep kan variere mye, og treningsettet kan være lite variert.

Testsettet benyttes til å verifisere kryssvalideringen, og vil konkret si hvor bra klassifisering vi ville oppnå ved å benytte vårt system i den perioden testeksemplene dekker.

Kapittel 7

Resultater

Dette kapitlet presenterer de viktigste resultatene av tester på den naive bayesianske klassifikatoren basert på et treningssett utarbeidet som i kapittel 5 og implementasjonen forklart i kapittel 6. Hensikten med testene er å kartlegge hvor mange riktige klassifiseringer som gjøres på hver av klassene «Falsk Alarm» og «Reell Alarm». Resultatene herfra legger grunnlaget for neste kapittel hvor vi blant annet diskuterer nytteverdien av et slikt høyere ordens IDS.

Kappitlet starter med å forklare bakgrunnsinformasjon for testene der vi redgjør for hvordan vi har kommet fram til resultatene. Deretter kommer resultatene, som er basert pp tester ved kryssvalidering og uavhengig testsett (testfunksjoner beskrevet i kapittel 6.5). Vi vil også se på hvordan vi kan påvirke resultatet ved å benytte sikkerhetsmargin (som beskrevet i kapittel 6.3.2), samt ved å benytte teknikken som vi kaller underklassifisering (kapittel 6.4).

7.1 Bakgrunnsinformasjon for resultater

Testing utføres ved å trene opp en naiv Bayes klassifikator på vårt spesifikke treningssett og så la denne klassifisere hver enkelt Snort-alarm i et definert testsett. Snort-alarmene vi skal bruke i testingen må være manuelt klassifisert på forhånd på samme måte som de i treningssettet. Dermed har vi en fasit som gjør oss i stand til å avgjøre om alarmene blir klassifisert riktig.

I alle tester benytter vi samme treningsett for å lære opp klassifikatoren[1]. Vi har som nevnt to testmetoder, kryssvalidering og uavhengig testsett. Vi vil se litt nærmere på disse testene og komme med statistisk data over hvor mange enkelte Snort-alarmer som blir testet ved hver av

[1] Med unntak ved underklassifisering der det orginale treningsettet blir modifisert før trening

testmetodene.

7.1.1 Bruk av kryssvalidering

Vi har utført kryssvalidering ved å tilfeldig dele opp treningssettet i ti like store deler. En av disse delene blir benyttet til testing og de resterende delene til trening. Dette gjentas ti ganger, slik at hver del er benyttet til testing en gang. I sum blir altså alle Snort-alarmer i treningsettet benyttet til testing.

Snort-alarmer totalt	20866
Forekomster av klassen «Falsk Alarm»	16405
Forekomster av klassen «Reell Alarm»	4461

Tabell 7.1: Størrelse og klassefordeling til treningssettet

Tabell 7.1 viser antall Snort-alarmer totalt i treningssettet, og klassefordelingen av disse. Dette betyr at kryssvalidering vil totalt klassifisere 20866 Snort-alarmer.

Kryssvalidering vil gi avvik i resultater om testen gjentas flere ganger fordi oppdelingen av treningssettet gjøres tilfeldig. Derfor er alle resultater av kryssvalideringstester vi presenterer et gjennomsnitt av av ti kryssvalideringer. Unntaket er om vi oppgir noe annet.

7.1.2 Bruk av uavhengig testsett

Med uavhengig testsett mener vi her Snort-alarmer som ikke inngår i treningssettet og bare blir benyttet til testing. En viktig forskjell fra kryssvalidering er at disse Snort-alarmene også stammer fra en annet tidsrom[2] enn de som er representert i treningssettet.

Snort-alarmer totalt	7425
Forekomster av klassen «Falsk Alarm»	2997
Forekomster av klassen «Reell Alarm»	4428

Tabell 7.2: Størrelsen og klassefordelingen i testsett

Som vi ser av tabell 7.2 vil vi klassifisere 7425 Snort-alarmer ved denne testmetoden.

[2]Dataene vi jobber med er fra februar og mars 2004. 14 dagers Snort-alarmer til treningssett og 3 påfølgende dagers alarmer til uavhengig testsett.

7.1.3 Presentasjon av resultater

Vi presenterer resultatene i et fast tabellformat. Tabellene er delt i to der øverste del viser de spesifikke resultatene fra klassifiseringen. Den nederste delen fremhever de to viktigste poengene ved testen. Dette er hvor mange falske alarmer filteret ville forhindret, og hvor mange reelle alarmer som ville blitt oversett om klassifikatoren ble benyttet som et filter i et «høyere ordens IDS». Den nederste delen av tabellen inneholder derfor ikke noen nye resultater, men hjelper til med å tolke hva resultatene vil si ved praktisk bruk av klassifisatoren.

- **Snort-alarmer totalt** – Her vises hvor mange Snort-alarmer som ble riktig klassifisert av alle som er i testsettet. Dette gir en indikasjon på hvor god klassifiseringen er totalt sett. Men, resultatet her vil styres mest av hvor bra klassifiseringen blir for klassen det finnes flest forekomster av i testsettet. Er det stor ulikhet mellom forekomster av klassene vil derfor ikke dette resultatet nødvendigvis gi riktig bilde av klassifiseringen. I vårt tilfelle vil dette gjelde kryssvalidering som vi kan se av tabell 7.1.

- **Feil klassifiseringer av klassen «Falsk Alarm»** – Her angis hvor mange Snort-alarmer av klassen «Falsk Alarm» som blir klassifisert galt.

- **Feil klassifiseringer av klassen «Reell Alarm»** – Her angis hvor mange Snort-alarmer med klassen «Reell Alarm» som blir klassifisert galt.

- **Gjennomsnittlig feilprosent mellom klassene** – Gjennomsnittet av prosentene fra de to foregående punkter. Som mål på hvor bra klassifiseringen er totalt sett er denne verdien bedre enn å se på riktig klassifiseringer totalt da denne ikke vil påvirkes av antall forekomster av klassene.

- **Usikre klassifiseringer** – Usikre klassifiseringer som følge av at en sikkerhetsmargin er angitt til klassifikatoren. Hensikten med de usikre klassifiseringene er at disse skal behandles på lik linje som reelle alarmer, men for tydelig å se effekten av sikkerhetsmarginen vises de her.

- **Reduksjon av falske alarmer** – Alle Snort-alarmer med klassen «Falske Alarmer» som klassifiseres riktig vil indikere en falsk alarm som ble stoppet om klassifikatoren ble benyttet som et filter. Dette punket er altså korrekt klassifiserte «Falske Alarmer», og er med for å fremheve dette poenget spesielt.

- **Reelle alarmer som oversees** – Samme som *Feil klassifiseringer av klassen «Reell Alarm»*. Nevnes for å fremheve poenget med at dette er reelle alarmer som blir klassifisert som falske og ville derfor bli fjernet før manuell behandling.

7.2 Resultater ved kryssvalidering

Vi tester klassifiseringen av Snort-alarmer ved å trene en klassifikator på treningssettet og kryssvalidere. Resultat i egen tabell 7.3.

Riktig klassifiserte Snort-alarmer totalt	18252	87,5 %
Feil klassifiseringer av klassen «Falsk Alarm»	1990	12,1 %
Feil klassifiseringer av klassen «Reell Alarm»	623	14,0 %
Gjennomsnittlig feilprosent mellom klassene		13,0 %
Usikre klassifiseringer	0	
I praksis vil dette si:		
Reduksjon av falske alarmer		**87,9 %**
Reelle alarmer som oversees		**14,0 %**

Tabell 7.3: Kryssvalidering av ordinært treningssett.

7.3 Resultater ved uavhengig testsett

Vi tester klassifiseringen av Snort-alarmer ved å trene en klassifikator på treningssettet og teste på uavhengig testsett. Resultatene sees i tabell 7.4.

Riktig klassifiserte Snort-alarmer totalt	6803	91,6 %
Feil klassifiseringer av klassen «Falsk Alarm»	400	13,3 %
Feil klassifiseringer av klassen «Reell Alarm»	222	5,0 %
Gjennomsnittlig feilprosent mellom klassene		9,2 %
Usikkre klassifiseringer	0	
I praksis vil dette si:		
Reduksjon av falske alarmer		**86,7 %**
Reelle alarmer som oversees		**5,0 %**

Tabell 7.4: Resultater med uavhengig testsett

7.4 Tester med sikkerhetsmargin

Vi utførte tester der vi benyttet muligheten for å sette sikkerhetsmargin i klassifikatoren. Vi husker fra kapittel 6.3.2 at denne angir hvor sikker klassifikatoren må være før den klassifiserer en Snort-alarm. Sikkerheten til en klassifisering beregnes utfra forholdet mellom sannsynlighetene for de to klassene. Er ikke klassifikatoren sikker nok, resulterer dette i en usikker klassifisering. Hensikten med sikkerhetsmargien er å minske feil klassifiseringer av klassen «Reell Alarm», slik at usikre klassifiseringer vil bli behandlet som «Reell Alarm». Derfor vil alarmer med riktig klass «Falsk Alarm» som får usikker klassifisering, regnes som feil klassifisert. Alarmer med klassen «Reell Alarm» som blir klassifisert som usikker, regnes som korrekt.

Vi tester ut ulike sikkerhetsmarginer, både ved kryssvalidering og ved uavhengig testsett. Sikkerhetsmarginen ble testet for alle elementer i mengden [10, 20, 30, 40, 50, 60, 70, 80, 85, 90, 95, 99, 99.9, 99.99, 99.999, 99.9999, 99.99999, 99.999999]. Vi presenterer bare de mest interessante resultatene her. Resultater for alle sikkerhetsmarginer finnes forøvrig i vedlegg B.

7.4.1 Sikkerhetsmargin ved kryssvalidering

Disse resultatene er et gjennomsnitt av fem kryssvalideringer. Se tabellene 7.5-7.8

Riktig klassifiserte Snort-alarmer totalt	17922	85,9 %
Feil klassifiseringer av klassen «Falsk Alarm»	2259	13,8
Feil klassifiseringer av klassen «Reell Alarm»	557	12,5 %
Gjennomsnittlig feilprosent mellom klassene		-
Usikre klassifiseringer	639	3,1 %
I praksis vil dette si:		
Reduksjon av falske alarmer		**86,2 %**
Reelle alarmer som oversees		**12,5 %**

Tabell 7.5: Kryssvalidering med sikkerhetsmargin = 50 %

Riktig klassifiserte Snort-alarmer totalt	16507	79,1 %
Feil klassifiseringer av klassen «Falsk Alarm»	3492	20,9
Feil klassifiseringer av klassen «Reell Alarm»	363	8,1 %
Gjennomsnittlig feilprosent mellom klassene		-
Usikre klassifiseringer	2679	12,8 %
I praksis vil dette si:		
Reduksjon av falske alarmer		**78,7 %**
Reelle alarmer som oversees		**8,1 %**

Tabell 7.6: Kryssvalidering med sikkerhetsmargin = 95 %

Riktig klassifiserte Snort-alarmer totalt	11562	55,4 %
Feil klassifiseringer av klassen «Falsk Alarm»	7897	48,1
Feil klassifiseringer av klassen «Reell Alarm»	62	1,4 %
Gjennomsnittlig feilprosent mellom klassene		-
Usikre klassifiseringer	8551	41,0 %
I praksis vil dette si:		
Reduksjon av falske alarmer		**51,9 %**
Reelle alarmer som oversees		**1,4 %**

Tabell 7.7: Kryssvalidering med sikkerhetsmargin = 99,99 %

Riktig klassifiserte Snort-alarmer totalt	10043	48,1 %
Feil klassifiseringer av klassen «Falsk Alarm»	8928	54,4
Feil klassifiseringer av klassen «Reell Alarm»	39	0,9 %
Gjennomsnittlig feilprosent mellom klassene		-
Usikre klassifiseringer	10312	49,4 %
I praksis vil dette si:		
Reduksjon av falske alarmer		**45,6 %**
Reelle alarmer som oversees		**0,9 %**

Tabell 7.8: Kryssvalidering med sikkerhetsmargin = 99,999 %

7.4.2 Sikkerhetsmargin på uavhengig testsett

Testene viser resultatet av klassifiseringen på uavhengig testsettet ved bruk av sikkerhetsmargin i klassifikatoren trent på vårt treningsett. (Tabell 7.9 og 7.10).

Riktig klassifiserte Snort-alarmer totalt	5263	71 %
Feil klassifiseringer av klassen «Falsk Alarm»	1556	51,9
Feil klassifiseringer av klassen «Reell Alarm»	57	1,3 %
Gjennomsnittlig feilprosent mellom klassene		-
Usikre klassifiseringer	1951	26,3 %
I praksis vil dette si:		
Reduksjon av falske alarmer		**48,1 %**
Reelle alarmer som oversees		**1,3 %**

Tabell 7.9: Test ved uavhengig testsett med sikkerhetsmargin = 99 %

Riktig klassifiserte Snort-alarmer totalt	4439	59,8 %
Feil klassifiseringer av klassen «Falsk Alarm»	1836	61,3
Feil klassifiseringer av klassen «Reell Alarm»	43	1,0 %
Gjennomsnittlig feilprosent mellom klassene		-
Usikre klassifiseringer	2855	38,5 %
I praksis vil dette si:		
Reduksjon av falske alarmer		**38,7 %**
Reelle alarmer som oversees		**1,0 %**

Tabell 7.10: Test ved uavhengig testsett med sikkerhetsmargin = 99,99 %

7.5 Underklassifisering

Ved å innføre underklasser i treningsettet, slik vi forklarte i kapittel 6.4, håper vi på å forbedre klassifikatoren slik at den klassifiserer flere Snort-alarmer korrekt. VI benytter en automatisert funksjon for å finne en god forekomst av underklassifisert treningssett.

Funksjonen varierer nivå av underklasser fra to til åtte nivåer. Nivåer er antall underklasser som hver opprinnelig klasse deles opp i. Ti forekomster av underklassifiserte treningsset ble testet ved hvert nivå. Funksjonen tar vare på det beste underklassifiserte treningssettet.

7.5.1 Kryssvalidering

Tabell 7.11 viser resultat av klassifisering av Snort-alarmer ved å trene opp klassifikator på underklassifisert treningsett og kryssvalidere.

Riktig klassifiserte Snort-alarmer totalt	18219	87,3 %
Feil klassifiseringer av klassen «Falsk Alarm»	2104	12,8 %
Feil klassifiseringer av klassen «Reell Alarm»	543	12,2 %
Gjennomsnittlig feilprosent mellom klassene		12.5 %
Usikre klassifiseringer	0	
I praksis vil dette si:		
Reduksjon av falske alarmer		**87,2 %**
Reelle alarmer som oversees		**12,2 %**

Tabell 7.11: Underklassifisert treningssett - Kryssvalidering

7.5.2 Resultater med uavhengig testsett

Denne testen viser resultater av klassifisering av Snort-alarmene i det uavhengige testsettet, ved å trene klassifikatoren på underklassifisert treningsett. (tabell 7.12).

Riktig klassifiserte Snort-alarmer totalt	6796	91,5 %
Feil klassifiseringer av klassen «Falsk Alarm»	378	12,6 %
Feil klassifiseringer av klassen «Reell Alarm»	251	5,7 %
Gjennomsnittlig feilprosent mellom klassene		9,2 %
Usikre klassifiseringer	0	
I praksis vil dette si:		
Reduksjon av falske alarmer		**87,4 %**
Reelle alarmer som oversees		**5,7 %**

Tabell 7.12: Underklassifisering av uavhengig testsett

7.6 Øvrige tester

I dette kapitlet har vi nøyd oss med å presentere de mest interessante resultatene. Som nevnt bestod testen av sikkerhetsmargin av flere deler enn

de som er vist. Vi har også testet sikkerhetsmargin ved både kryssvalidering og testsett på det underklassifisert treningssettet. For disse testene henviser vi til vedlegg B, side 77.

Kapittel 8

Diskusjon av hypoteser og resultater

Vi begynner dette kapitlet med å drøfte hvordan testene i forrige kapittel støtter opp om de tre hypotesene fremlagt i kap 1.

Vi vil deretter diskutere resultatene av testene mer inngående. Fokuset vil være rettet mot evnen til å klassifisere reelle alarmer riktig, samtidig som vi forsøker å redusere så mange falske alarmer som mulig. Vi kommer inn på hvorvidt bruk av teknikker som underklasser og sikkerhetsmargin er nyttig utfra resultatene vi fikk ved testing.

Tilslutt vil vi drøfte nytteverdien av et «høyere ordens IDS» (basert på naiv Bayes klassifikator) i forhold til ren manuell analyse. I denne sammenhengen vil vi også fremme forslag for hvordan systemet bør brukes basert på resultatene fra forrige kapittel.

8.1 Drøfting av hypoteser

Hypotese 1
Utsagnet i hypotese 1 gikk på hvorvidt maskinlæring som metode kan benyttes til å klassifisere alarmer fra signaturbaserte IDS. Utfra tabell 7.3 og 7.4 ser vi at nærmere 90 prosent av alle alarmer klassifisert riktig med hensyn på klassene «Falsk Alarm» og «Reell Alarm». Dette tyder på at Snort-alarmer er godt egnet til bruk i maskinlæring, og at det lar seg gjøre å klassifisere disse som reelle alarmer eller falske alarmer.

Hadde motsatt vært tilfelle ville vi fått en tilfeldig klassifisering av alarmer, slik at en da kunne forventes å ende opp rundt 50 prosent riktig klassifisering.

Siden Snort alarmer består av pakke header og payload fra internettprotokollene vil dette gjelde generellt for IDS-alarmer som kan består av dette. Dermed kan vi si at vi har funnet støtte for denne hypotesen.

Hypotese 2
Hypotese 2 sier at falske alarmer i stor grad kan reduseres ved bruk av en maskinlæringsmetode. De samme testene som støttet hypotese 1 vil også støtte hypotese 2. Vi ser av disse testene at falske alarmer ville kunne blitt redusert mellom 85 og 90 prosent om vi hadde filtrert bort alle alarmer som klassifiseres som falske. Dette vil vi betrakte som en betydelig reduksjon, slik at vi kan konkludere med at resultatene støtter hypotesen.

Hypotese 3
Utsagnet her sier vi kan benytte metoder for å minimere feilklassifisering av reelle alarmer og samtidig opprettholde en god klassifisering av falske alarmer.

Ved å sammenligne testene i tabell 7.3 med 7.8 og 7.4 med 7.9, ser vi hvordan en sikkerhetsmargin kan gjøre nettopp dette. I begge tilfeller reduseres feil klassifisering av reelle alarmer til omtrent 1 prosent, fra opprinnelig 14 og 5 prosent. Fortsatt vil nærmere 50 prosent av alle falske alarmer kunne forhindres og må derfor sies å fortsatt ha nytteverdi. Dermed har testene våre også gitt støtte for denne siste hypotesen.

8.2 Testresultater ved kryssvalidering og uavhengig testsett

8.2.1 Kryssvalidering

Ved kryssvalidering ser vi av tabell 7.3 at Snort-alarmer vil bli klassifisert riktig i totalt 87 prosent av tilfellene. Dette ser vi direkte av *riktige klassifiseringer totalt*.

Gjennomsnittlig feilprosent mellom klassene er 13 prosent, og tyde dermed også på at 87 prosent av Snort-alarmer blir riktig klassifisert.

Om klassifisatoren blir brukt som et filter ser vi at omtrent 88 prosent av falske alarmer vil reduseres. Dette er et resultat som utvilsomt ville lettet arbeidspesset på analytikere som behandler Snort-alarmer. Ulempen er at 14 prosent av reelle alarmer ville blitt oversett. Selv om angrep ofte utløser en rekke alarmer vil disse ofte være så like at hele angrepet står i fare for å bli oversett. For at klassifikatoren skal kunne brukes i praksis bør derfor evnen til å klassifisere reelle alarmer økes.

8.2.2 Uavhengig testsett

Når vi klassifiserer Snort-alarmene i vårt uavhengige testsett (tabell 7.4) blir *gjennomsnittlig feilprosent mellom klassene* 9,2 prosent. At resultatene varierer mellom denne testen og kryssvalidering er en naturlig følge av at

dette er Snort-alarmer fra forskjellige tidsperioder. Likevel ser vi den gjennomsnittlig feilprosent ikke spriker mer enn 3,8 prosent. Siden resultatene ligger såpass nærme hverandre, er dette et tegn på at testene gir et realistisk bilde på hvor god klassifiseringen av Snort-alarmer blir ved å trene en klassifikator på vårt treningsett. Hadde treningssettet vårt vært mer variert, eller inneholdt alarmer fra et større tidsrom, kunne forskjellen her blitt enda mindre.

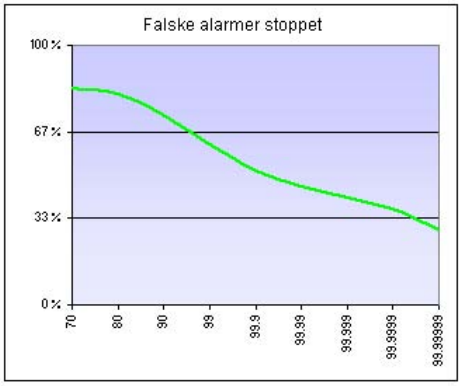
I forhold til kryssvalideringen ville reduksjonen av falske alarmer blitt noe mindre, mens nå bare 5 prosent av reelle alarmer blir oversett. Vi vil ikke dra noen sluttninger på disse ulikhetene da dette antagelig skyldes tilfeldigheter, som for eksempel at angrepene i testsettet er letterere å klassifisere enn alarmer i treningssettet. Ved andre testdata kunne det motsatte vært tilfelle.

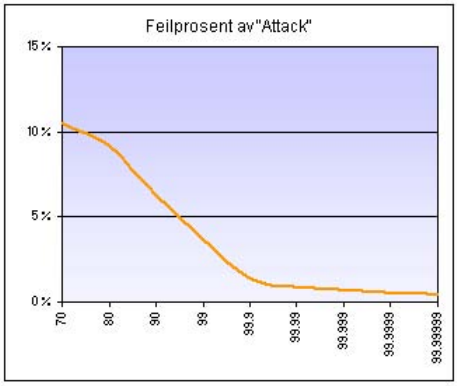
8.3 Sikkerhetsmargin

Ved å øke *sikkerhetsmarginen* ser vi at vi får en økende mengde usikre klassifiseringer. Dette kan vi se av resultatene i tabellene 7.5 til 7.8.

Tabell 7.5 viser resultatet av kryssvalidering når vi setter en margin på 50 prosent til klassifikatoren. Vi ser her at ca 3 prosent av klassifiseringene blir usikre. Ved å øke *sikkerhetsmarginen* til 99,999 prosent blir nærmere 50 prosent av alle klassifiseringer usikre.

Grunnen til usikre klassifiseringer er at vi nå ikke klassifiserer Snort-alarmer med mindre klassen kan angis med større sikkerhet enn hva *sikkerhetsmarginen* tilsier. Dette gir en mulighet til å overstyre disse til å gå til manuell analyse. Usikre klassifiserte Snort-alarmer blir derfor behandlet på lik linje som Snort-alarmer klassifisert som «Reell Alarm».





Grafene over er generert fra kryssvaliderings-tester av en klassifikator der vi har innført *sikkerhetsmargin*. Grafen til venstre illustrerer hvordan færre og færre falske alarmer forhindres av filtere når sikkerhetsmarginen

økes. Grafen til høyre viser hvordan reelle alarmer som oversees minker når *sikkerhetsmarginen* øker. Førsteaksen i begge grafene representerer sikkerhetsmarginen. Når *sikkerhetsmarginen* er 99 eller større blir denne aksen logaritmisk for å få frem de plutselige forandringen i grafen.

Det første vi vil kommentere er hvordan *sikkerhetsmarginen* plutselig vil få stor effekt ved ørsmå endringer når den kommer til et vist nivå. Dette taes i betraktining når *sikkerhetsmarginen* skal bestemmes. I vårt tester fungerte det fint med en trinnvis økning av *sikkerhetsmarginen* med $\frac{1}{10}$ av forrige økning.

Av grafene ser vi falske alarmer som stoppes av filteret vil reduseres jevnt og trutt når vi øker sikkerhetsmarginen. Reelle alarmer som overses vil minke en stund, men tilslutt komme til et punkt der videre økning av *sikkerhetsmarginen* vil ha liten effekt. Dette punktet ligger for *sikkerhetsmargin* mellom 99,9 og 99,99 prosent i vårt tilfelle.

Selv om man ønsker å ha en minimal feilklassifisering av klassen «Reell Alarm», vil det være liten hensikt å øke *sikkerhetsmarginen* over dette punktet, da dette ikke gjør annet en å minske evnen til å oppdage falske alarmer.

Resultatene av *sikkerhetsmarginen* varierer ved kryssvalidering og ved uavhengig testsettet (tabell 7.9 og 7.10). Test med uavhengig testsett får vi langt raskere flere usikre klassifiseringer ved å øke *sikkerhetsmarginen*. Dette betyr at klassifikatoren viser større usikkerhet når vi tester på Snort-alarmer fra en annen tidsperiode enn de som finnes i treningsettet.

Ser vi på hvilke resultater vi oppnår, er fordelen med å innføre sikkerhetsmargin klare. I utgangspunktet ville såpass mange reelle alarmer oversees av klassifikatoren at vi kunne sette spørsmålstegn ved nytteverdien til denne i et «høyere orden IDS». Ved å innføre sikkerhetsmargin ser vi at feil klassifiseringen av reelle alarmer kan presses ned mot 1 prosent, og likevel ha en god effekt da omlag 50 prosent av falske alarmer blir forhindret (tabell 7.8 og 7.9). Dette ville gjort hverdagen til analysepersonell langt lettere.

Da våre to testmetoder ga svært ulik resultat for å oppnå denne effekten bør *sikkerhetsmargin* testes grundig før den bestemmes. Vi setter mest lit til testen på uavhengig testsett, da klassifikatoren i praksis vil møte Snort-alarmer fra tidsperioden etter treningsettet. Det kan være lurt å sette en høy *sikkerhetsmargin* og heller justere denne etterhvert.

8.4 Underklassifisering

Vi må gå utifra vi ikke har funnet beste forekomst av underklassifisert treningssett ved våre tester. Leting etter slike er en svært tidkrevende prosess, og vi antar at brukerne av systemet vil ha en tidsbegrensning på denne prosessen. Våre resultater er derfor forholdsvis realistisk med tanke på dette.

Vi vil sammenligne de konkrete resultater for kryssvalideringstest på en klassifikator trent på underklassifisert treningssett (tabell 7.11) mot det «ordinære» treningssett (tabell 7.3). Vi ser en liten reduksjon av gjennomsnittlig feilprosent med 0,5 prosent. Feil klassifisering av klassen «Reell Alarm» har blitt bedret med 1,8 prosent, mens feil klassifiseringen av falske alarmer er gått opp 0,7 prosent. At riktige klassifiserte Snort-alarmer totalt har økt noe totalt sett ser vi på som en følge at klassen «Falsk Alarm» har langt flere forekomster enn klassen «Reell Alarm» i treningsettet.

Ved testing av klassifisatoren på uavhengig testsett (7.4 og 7.12) ser vi at resultatene blir påvirket i enda mindre grad. I dette tilfellet vil ikke gjennomsnittelig feilklassifisering mellom klassene forandre seg.

Disse resultatene tyder på en veldig liten forbedring av klassifiseringen ved å benytte et underklassifisert treningssett. Dette svarer ikke helt til forventningene, da vi på forhånd trodde denne teknikken skulle bidra i større grad til å minske feil klassifiseringer på begge klasser. Det kan skyldes flere grunner, men vi mener to teorierer utpeker seg:

- Underklassifisering egner seg av en eller annen grunn ikke til bruk på Snort-alarmer.
- Inkonsistent klassifisering og feilkilder i treningssettet (kap 5.6) gjør at enkelte av underklassene i «Reell Alarm» og «Falsk Alarm» kan bli svært like. Vi tenker oss et ekstremt tilfelle der en type Snort-Alarm finnes for begge klasser pga feilklassifisering: *Om det finnes mange av denne Snort-alarmen kunne det tenkes at den dannet en egen underklasse for begge de opprinnelige klassene*. Dermed ville vi fått to underklasser som var svært like, som uten tvil ville føre til feil.

Av disse to er det feilkilder i treningssettet som kan gjøres noe med. Vi mener dette kan være mye av grunnen til at underklassifisering ikke gir oss effekten vi ønsker. Skulle man klare å frembringe et mer støyfritt treningssett vil underklassifisering kreve ytterligere testing før en kan komme med bastante slutninger om nytteverdien av underklassifisering. Vi står derfor igjen kun med antagelsen om at underklassifisering er en potensielt god metode for å bedre klasifiseringen av Snort-alarmer. Vi vil derfor ikke anbefale å benytte denne teknikken ved vårt forslag for et «høyere ordens IDS».

8.5 Nytteverdien av vårt «høyere ordens IDS»

Ved å benytte en høy *sikkerhetsmargin* har «naiv Bayes klassifikator» som klassifiseringsteknologi gitt oss gode resultater på å klassifisere Snort-alarmer som falsk eller reell alarm. Antall falske alarmer kan bli redusert i stor grad uten at man overser mange reelle alarmer. Å bruke dette i kombinasjon

med manuell logging vil da kunne lette trykket på en analytiker. I Telenor-caset vil dette kunne øke kapasiteten ved sikkerhetssenteret uten økt brukt av ressurser.

Våre resultater er som nevnt preget av enkelte problemer med støy og forholdsvis lite variasjon treningssett (kap 5.6). Dette er problemer det til en viss grad kan gjøres noe med, noe vi kommer med konkrete forslag til i neste kapittel. Derfor mener vi at klassifiseringen av Snort-alarmer kan forbedres i forhold til våre resultrater.

Ved manuell behandling av alarmer vil det alltid være fare for gjøre feil. Et høyere ordens IDS som presentert her, vil kunne redusere strømmen av falske alarmer mot analytiker, slik at denne feilkilden vil minke. Analytiker får også større mulighet til å oppdage alvorlige angrep tidligere slik at disse kan kan behandles raskere. Den totale effektiviteten til vårt «høyere ordens IDS» kan derfor vise seg å være like god, men med langt mindre arbeid for en analytiker.

Likevel ser vi enkelte farer med et slikt system. Vi har i utgangspunktet ingen kontroll over hvilke angrep de reelle alarmer som bli klassifisert som falsk alarm representerer. Dette kan like gjerne hende svært alvorlig angrep som noe annet.

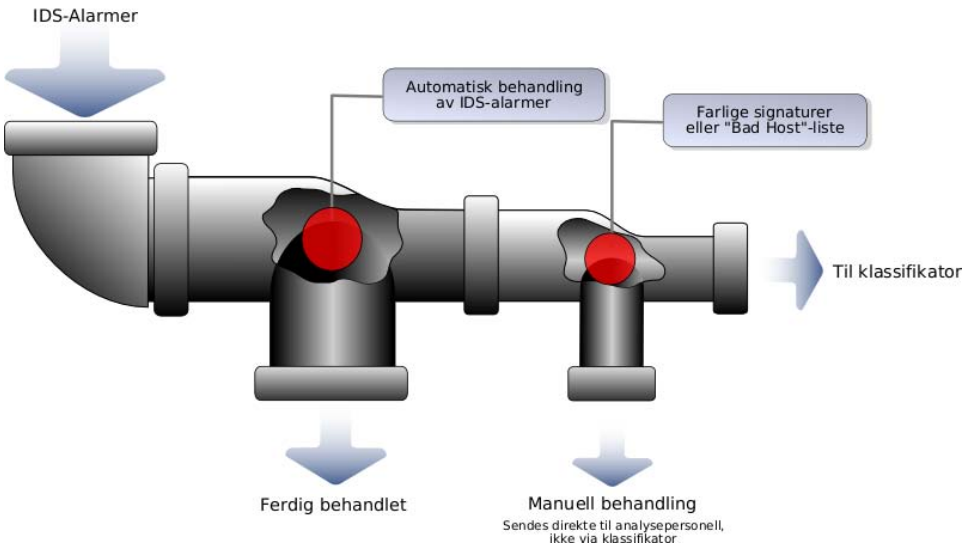
Vi kommer her med konkrete forslag for rutiner og metoder som vi mener bør inngå som en naturlig del av ett «høyere ordens IDS». Forslagene er i stor grad bestemt av resultatene vi har kommet frem til ved testingen i forrige kapittel.

Bruk av sikkerhetsmargin
Sikkerhetsmargin ga ønsket effekt på systemet, og vil anbefales å benyttes ved et «høyere ordens IDS» basert på naiv Bayes klassifikator

Bruke klassifikatoren som rådgiver
Før man tar et slikt filter i bruk, lar man klassifikatoren heller foreslå klassifisering til analysepersonell over en tidsperiode. Hensikten er å få et innblikk i hvor god klassifiseringen er i praksis, samt kunne finjustere sikkerhetsmarginen etter ønsket effekt.

Overstyre alarmer
Vi vil anbefale en mulighet til å navigere Snort-alarmer utenom klassifikatoren. Dette kan baseres på speielle signaturer, IP adresser (bad host liste) eller lignende. Dette gir mulighet til tvinge spesielt farlige angrep/signaturer til alltid å gå til manuell behandling. Det viser seg at alvorlige angrep er forholdsvis sjeldne, derfor vil dette ikke redusere et slikt filters effektivitet i merkbar grad.

I tilfeller der Snort-alarmer av en spesiell signatur alltid kan behandles likt, vil vi også anbefale at disse blir automatisk behandlet uten hverken å gå til filteret eller manuell behandling. En modul for dette finnes i vår case med Telenor, og fungerer svært godt sammen med en naiv Bayes klassifikator i et «høyere ordens IDS».



Figur 8.1: Figuren illustrerer hvordan vi kan sile ut enkelte Snort-alarmer etter ønske, slik at disse ikke går til klassifikatoren (filteret). Dette gjør systemet mer dynamisk.

Oppdatering av treningssett

Klassifikatoren er avhengig av gode rutiner for å oppdatere treningssett for å opprettholde effektivitet. Dette vil øke usikre klassifiseringer, om sikkerhetsmargin benyttes, slik at flere falske alarmer blir fjernet. Utdaterte treningssett vil derfor ikke først og fremst overse reelle alarmer, men færre falske alarmer vil stoppes.

Ved å benytte verktøy som automatisk lager et treningssett etter retningslinjene i kapittel 5 er dette en enkel prosess. Så sant ikke underklassifisering er tatt i bruk, vil det gå kjapt både å lage treningssett og benytte dette i læringsprosessen (trene opp klassifikatoren). Ved underklasser må vi derimot finne en god forekomst av underklassifisert treningssett, noe som kan være tidkrevende.

Hvor ofte det er nødvendig å oppdatere treningssettet er noe som kan undersøkes nærmere i perioden klassifikatoren benyttes som rådgiver.

Kapittel 9

Mulige forbedringer

Vår implementasjon og testing av «høyere ordens IDS» kan sees på som en tidlig løsning som har potensiale for å kunne effektiviseres. Vi vil her fremme forslag for metoder til å øke riktig klassifiseringer, samt noen strategier for å videreutvikle systemet. Øke riktig klassifiseringer vil bestå i metoder å forbedre treningssettet.

9.1 Forbedre treningsettet

At treffsikkerheten til klassifikatoren ikke er høyere en resultatene viser, bør bety rom for forbedringer. Først og fremst ligger mye av dette i treningssettet, men også andre tiltak kan være nyttig. Her kommer noen enkle anbefalinger for å øke ytelsen til klassifikatoren.

La all treningseksempler være klassifisert
La alle Snort-alarmer bli klassifisert ved manuell behandling og ikke bare en enkelt alarm slik situasjonen er i vårt tilfelle (kap 1.4). Selv om dette er et case-relatert problem , kan det tenkes andre benytter lignende klassifiseringsrutiner.

Større og mer variert treningssett
Treningssettet vil da presentere bredden av alarmene bedre. En strategi for å variere treningssettet kan være å sørge for at en gitt mengde Snort-alarmer for hver signatur som representere båre reelle og falske alarmer. Dette kan vise seg å minke variasjonen i resultater mellom de enkelte periodene.

Klassifikatoren som rådgiver
Dette tiltaket har vi nevnt i forrige kapittel, men passer også her da dette vil bekjempe inkonsistent logging. Det skjer fordi forslaget til klasse vil være med på å påvirke analysepersonell når det oppstår tvilstilfeller.

9.2 Videreutvikling av systemet

9.2.1 Sanntidslæring

En spennende tanke er et «høyere ordens IDS» som oppdaterer seg selv i sanntid ved hjelp av innspill fra analysepersonell. Da naiv Bayes læringsmetode består av enkelt tallmateriale vil det være en enkelt å oppdatere dette, ved å addere med en, for hver gang man ser en attributtverdi og klasse. Om dette implementeres kan behovet for å designe treningssett etterhvert forsvinne.

En mulighet her er at alle Snort-alarmer som går til manuell analyse, som følge av «usikker» klassifisering, fører til oppdatering i læringsmaterialet når denne alarmen blir klassifisert manuelt. I vår implementasjon ligger både muligheten for å oppdatere læringsmaterialet samt funksjonalitet for å lagre dette til database. Det som gjenstår er koblingen mot den manuelle loggingen.

9.2.2 Alternative klasser

Noe som er naturlig å jobbe videre med er å vurdere alternative måter å klassifisere alarmer på, for så å teste dette.

Dekker innbruddsdeteksjonsystemet flere nettverk, som i vår case, kan det forekomme forskjeller i både klassifiseringsrutiner og typer nettverkstrafikk. Dette kan motivere til å lage egne treningssett for hvert nettverk, slik at vi kan trene opp en klassifikator for hvert nettverk.

En annen ide for alternativ klassifisering, er å ha en to klasser for hver signatur. Disse må representere falsk og reell alarm for signaturene. To muligheter åpner seg: Et treningssett kan lages for hver signatur, for å trene opp signaturspesialiserte klassifikatorer. Ved å se hvilken signatur en alarm har utløst, kan alarmen sendes til riktig klassifikatoren. Ulempen er at dette krever at svært mange klassifikatorer må behandles parallelt. Alternativet er å lage ett treningssett og trene en «universal» klassifikator på dette. Ved å se om alarmer blir klassifisert som reell/falsk for samme signatur-klasse som den faktiske signaturen som utløste alarmen, kan vi få en form for kvalitetssikring av klassifikatoren. Hvis dette ikke stemmer overens, er alarmen feilklassifisert.

Kapittel 10

Konklusjon

Signaturbaserte innbruddsdeteksjonssystemer er et viktig verktøy for å oppdage angrep og uønsket nettverkstrafikk. Et problem med disse er at det genereres mange falske alarmer. Falske alarmer skaper press på analysepersonell, og er ressurskrevende.

Vi har presentert en løsning på dette som vi kaller «høyere ordens IDS». Her kombineres Snort, som signaturbasert IDS, med maskinlæringsmetoden naiv Bayesiansk klassifikator.

Hensikten er å la den naive Bayesiske klassifikator læres opp til å skille false alarmer fra reelle, slik at denne kan benyttes som filter for å redusere falske alarmer i et signaturbasert IDS. Filteret må kun fjerne en liten mengde av reelle alarmer om systemet skal kunne brukes.

Tester viser at vårt «høyere ordens IDS» vil kunne reduseret falske alarmer med omlag 50 prosent ved kun å stoppe rundt 1 prosent av reelle angrep.

Dette vil i stor grad lette press på analytiker, og med hensyn på dette må resultatene sies å være en syksess. Skadeeffekten av at enkelte reelle alarmer også vil bli stoppet i filteret kan taes hensyn til ved la spesiellt viktige signaturer el gå utenom filteret, slik at disse ikke kan fjernes her.

Referanser

[1] Beale, Foster, Posluns, og Caswell. *Snort 2.0 Intrusion Detection*. Syngress Publishing, Inc., 2003.

[2] Bulatovic og Velasevic. A distributed intrusion detecion system based on bayesian alarm networks. Fra *Proceedings of the Secure Networking - 99 Conference*, Düsseldorf, nov/dec 1999.

[3] Darpa. `http://www.darpa.mil/`.

[4] Daejoon Joo, Taeho Hong, og Ingoo Han. The neural network models for ids based on the asymmetric costs of false negative errors and false positive errors. *Expert Systems with Applications*, 25(1), July 2003.

[5] Latex. `http://www.latex-project.org`.

[6] Matthew V. Mahoney. Network traffic anomaly detection based on packet bytes. Fra *Proceedings of the 2003 ACM symposium on Applied computing*, side 346–350. ACM Press, 2003.

[7] McClure, Scambray, og Kurtz. *Hacking Exposed: Network Security Secrets & Solutions, Fourth Edition*. McGraw Hill / Osborn, 2003.

[8] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[9] Nimda virus profile. `http://us.mcafee.com/virusInfo/default.asp?id=description&virus_k=99209`.

[10] Steven L. Scott. A bayesian paradigm for designing intrusion detection system. *Computational Statistics and Data Analysis*, 45(1), side 69–83, feb 2004.

[11] Sqlite. `http://sqlite.sourceforge.net/`.

[12] T. Verwoerd og R. Hunt. Intrusion detection techniques and approaches. *Computer Communications*, 25(15), side 1356, sep 2002.

[13] Vigna, Jonsson, og Kruegel. *Recent advancens in intrusion detection*. Springer, 2003.

Vedlegg A

Test på minsteverdi

A.1 Test 1

Testen viser resultater i klassifiseringen ved kryssvalidering av ordinært treningsett, når minsteverdien settes til en fast verdi. De ulike resultatene for minsteverdi presenterer et gjennomsnitt av ti kryssvaliderte tester.

Kolonnene har føgende verdier: Minimumsverdi, totalt korrekte klassifiseringer, totalt feil klassifiseringer, feilklassifiserte «Reelle Angrep», feilklassifiserte «Falske Alarmer»

Min. Verdi	Riktig klasse.	Galt klasse.	Feil i «Reelle Angrep»		Feil i «Falsk Alarm»	
0,01	14534	6333	792	17,76 %	10865	33,77 %
0,001	17389	3477	658	14,74 %	13586	17,19 %
1,5E-4	18395	2471	646	14,49 %	14581	11,12 %
1,4E-4	18414	2452	646	14,47 %	14599	11,01 %
1,3E-4	18437	2429	647	14,49 %	14623	10,86 %
1,2E-4	18444	2422	645	14,46 %	14628	10,83 %
1,1E-4	18472	2395	641	14,36 %	14651	10,69 %
1E-4	18488	2378	641	14,36 %	14667	10,59 %
9E-5	18510	2357	635	14,24 %	14684	10,49 %

Tabellen fortsett på neste side.

Min. Verdi	Riktig klasse.	Galt klasse.	Feil i «Reellt Angrep»		Feil i «Falsk Alarm»	
8E-5	18523	2343	637	14,27 %	14699	10,40 %
7E-5	18544	2322	637	14,28 %	14720	10,27 %
6E-5	18567	2299	633	14,20 %	14740	10,15 %
5E-5	18579	2287	633	14,18 %	14751	10,08 %
4E-5	18592	2274	633	14,19 %	14764	10,00 %
3E-5	18607	2260	634	14,21 %	14779	9,91 %
2E-5	18632	2234	631	14,14 %	14802	9,77 %
1E-5	18652	2214	634	14,20 %	14824	9,64 %
7,5E-6	18656	2210	634	14,21 %	14829	9,61 %
5E-6	18662	2204	635	14,23 %	14836	9,56 %
2,5E-6	18683	2183	637	14,27 %	14859	9,43 %
1,25E-6	18695	2171	643	14,40 %	14877	9,32 %
7,5E-7	18707	2159	646	14,48 %	14892	9,22 %
5E-7	18720	2146	647	14,49 %	14906	9,14 %
2,5E-7	18757	2109	650	14,58 %	14946	8,89 %
1,25E-7	18763	2103	658	14,74 %	14960	8,81 %
7,5E-8	18769	2097	661	14,81 %	14968	8,76 %
5E-8	18767	2099	664	14,89 %	14970	8,75 %
2,5E-8	18771	2095	673	15,08 %	14983	8,67 %
1,25E-8	18762	2104	686	15,38 %	14987	8,65 %
1E-14	18738	2128	722	16,18 %	14999	8,57 %
1E-22	18737	2129	724	16,22 %	14999	8,57 %

A.2 Test 2

Riktig klassefisert		Galt klassefisert		Feil i «Reellt Angrep»		Feil i «Falsk Alarm»	
18239	13,08 %	2627	12,59 %	622	13,9 %	14400	12,22 %
18229	13,25 %	2637	12,64 %	639	14,3 %	14407	12,18 %
18256	13,02 %	2610	12,51 %	621	13,9 %	14416	12,12 %
18286	12,91 %	2580	12,36 %	618	13,9 %	14443	11,96 %
18267	12,95 %	2599	12,46 %	616	13,8 %	14422	12,09 %
18243	13,14 %	2623	12,57 %	631	14,1 %	14413	12,14 %
18257	13,04 %	2609	12,50 %	623	14,0 %	14419	12,11 %
18262	12,98 %	2604	12,48 %	618	13,9 %	14419	12,11 %
18239	13,07 %	2627	12,59 %	620	13,9 %	14398	12,23 %
18260	12,99 %	2606	12,49 %	619	13,9 %	14418	12,11 %
18254	13,04 %	2612	12,52 %	623	13,96 %	14416	12,13 %

Tester resultater i klassifiseringen ved kryssvalidering av ordinært treningssett, når minsteverdien settes til $\frac{1}{|k|}$. Kryssvalidert 10 ganger. Nederste linje i tabellen viser gjennomsnittelig resultat. Kolonnene har føgende verdier: Totalt korrekte klassifiseringer, totalt feil klassifiseringer, feilklassifiserte «Reelle Angrep», feilklassifiserte «Falske Alarmer»

Vedlegg B

Klassifiserings tester

B.1 Ordinært treningssett

B.1.1 Kryssvalidering av ordinært treningssett

Tester klassifikatorens ytelse ved å Kryssvaliderer ordinært treningssett. Her vises hvert delresultat, som får frem en liten variasjon ved hver kryss-valideringstest [1]. Testen gjentas derfor 10 ganger for så å presenterer en gjennomsnittlig ytelse av testene tilslutt.

Margin	Riktig klasse.	Galt klasse.	Usikre	Feil i «Reellt Angrep»		Feil i «Falsk Alarm»	
0	18263	2603	0	624	13,99 %	1979	9,48 %
0	18254	2612	0	619	13,88 %	1993	9,55 %
0	18251	2615	0	626	14,03 %	1989	9,53 %
0	18258	2608	0	606	13,58 %	2002	9,59 %
0	18257	2609	0	617	13,83 %	1992	9,55 %
0	18262	2604	0	622	13,94 %	1982	9,50 %
0	18254	2612	0	630	14,12 %	1982	9,50 %
0	18239	2627	0	630	14,12 %	1997	9,57 %
0	18253	2613	0	627	14,06 %	1986	9,52 %
0	18235	2631	0	630	14,12 %	2001	9,59 %
0	18252	2613	0	623	13,97 %	1990	9,54 %

[1] Variasjonen skyldes underklasse tildeles tilfeldig

B.1.2 Kryssvalidering av ordinært treningssett med sikkerhetsmargin

Innfører sikkerhetsmargin, og tester ytelsen til klassifikatoren ved kryssvalidering. Sikkerhetsmarginen varieres (oppgis ved hver test). Hver resultat er et gjennomsnitt av 5 kryssvalideringstester.

Margin	Riktig klasse.	Galt klasse.	Usikre	Feil i «Reellt Angrep»		Feil i «Falsk Alarm»	
10	18208	2562	94	610	13,67 %	1952	9,35 %
20	18156	2500	209	597	13,38 %	1903	9,12 %
30	18096	2455	314	591	13,25 %	1864	8,93 %
40	18006	2392	467	575	12,89 %	1817	8,71 %
50	17922	2304	639	557	12,49 %	1747	8,37 %
60	17823	2230	811	538	12,06 %	1692	8,11 %
70	17653	2130	1082	509	11,41 %	1621	7,77 %
80	17424	2008	1433	469	10,51 %	1539	7,38 %
85	17247	1925	1693	440	9,86 %	1485	7,12 %
90	16963	1814	2088	410	9,19 %	1404	6,73 %
95	16507	1679	2679	363	8,14 %	1316	6,31 %
99	15502	1429	3934	283	6,34 %	1146	5,49 %
99.9	13488	1066	6312	164	3,68 %	902	4,32 %
99.99	11563	751	8551	62	1,39 %	689	3,30 %
99.999	10043	510	10312	39	0,87 %	471	2,26 %
99.9999	9151	387	11327	32	0,72 %	355	1,70 %
99.99999	8293	311	12261	24	0,54 %	287	1,38 %
99.999999	6748	270	13847	21	0,47 %	249	1,19 %

B.1.3 Sikkerhetsmargin på testsett på ordinært treningssett

Tester ytelsen på klassifikatoren på reservert testsett. Innfører sikkerhetsmargin, som varieres, og oppgis ved hver deltest. Hver resultat er et gjennomsnitt av 5 kryssvalideringstester.

Margin	Riktig klasse.	Galt klasse.	Usikre	Feil i «Reellt Angrep»		Feil i «Falsk Alarm»	
0	6803	622	0	222	4,98 %	400	1,92 %
10	6796	602	27	221	4,95 %	381	1,83 %
20	6785	593	47	218	4,89 %	375	1,80 %
30	6778	578	69	207	4,64 %	371	1,78 %
40	6765	571	89	207	4,64 %	364	1,74 %
50	6756	566	103	206	4,62 %	360	1,73 %
60	6731	535	159	193	4,33 %	342	1,64 %
70	6691	503	231	175	3,92 %	328	1,57 %
80	6625	473	327	170	3,81 %	303	1,45 %
90	6428	379	618	107	2,40 %	272	1,30 %
99	5912	289	1224	81	1,82 %	208	1,00 %
99.9	5263	211	1951	57	1,28 %	154	0,74 %
99.99	4439	131	2855	43	0,96 %	88	0,42 %
99.9999	1877	24	5524	2	0,04 %	22	0,11 %
99.99999	1432	17	5976	2	0,04 %	15	0,07 %
99.999999	1097	17	6311	2	0,04 %	15	0,07 %

B.2 Underklassifisert treningssett

B.2.1 Underklassifisering - kryssvalidering - sikkerhetsmargin

Benytter kryssvalidering til å teste ytelsen til underklassifisert treningssett. Tester ved ulike sikkerhetsmarginer, som oppgis ved hvert resultat. Resultatene er et gjennomsnitt av 5 kryssvalideringstester.

Margin	Riktig klasse.	Galt klasse.	Usikre	Feil i «Reellt Angrep»		Feil i «Falsk Alarm»	
0	18229	2637	0	543	12,17 %	2093	10,03 %
10	18152	2596	118	534	11,97 %	2062	9,88 %
20	18081	2535	250	523	11,72 %	2012	9,64 %
30	17969	2494	403	514	11,52 %	1980	9,49 %
40	17891	2406	569	485	10,87 %	1921	9,21 %
50	17754	2329	783	468	10,49 %	1861	8,92 %
60	17596	2246	1024	453	10,15 %	1793	8,59 %
70	17352	2147	1367	432	9,68 %	1715	8,22 %
80	17020	2010	1836	404	9,06 %	1606	7,70 %
85	16761	1933	2172	388	8,70 %	1545	7,40 %
90	16382	1820	2663	363	8,14 %	1457	6,98 %
95	15793	1635	3438	330	7,40 %	1305	6,25 %
99	14264	1402	5199	283	6,34 %	1119	5,36 %
99.9	12269	1157	7440	217	4,86 %	940	4,50 %
99.99	10304	874	9688	93	2,08 %	781	3,74 %
99.999	8819	715	11332	53	1,19 %	662	3,17 %
99.9999	7667	487	12712	42	0,94 %	445	2,13 %
99.99999	6555	284	14027	32	0,72 %	252	1,21 %
99.999999	5248	203	15414	20	0,45 %	183	0,88 %

B.2.2 Underklassifisering - testsett - sikkerhetsmargin

Benytter reservert testsett til å teste ytelsen til underklassifisert treningssett. Tester ved ulike sikkerhetsmarginer, som oppgis ved hvert resultat. Resultatene er et gjennomsnitt av 5 kryssvalideringstester.

Margin	Riktig klasse.	Galt klasse.	Usikre	Feil i «Reellt Angrep»		Feil i «Falsk Alarm»	
0	6796	629	0	251	5,63 %	378	1,81 %
10	6780	617	28	245	5,49 %	372	1,78 %
20	6739	603	83	239	5,36 %	364	1,74 %
30	6705	592	128	235	5,27 %	357	1,71 %
40	6687	577	161	228	5,11 %	349	1,67 %
50	6662	569	194	228	5,11 %	341	1,63 %
60	6579	549	297	218	4,89 %	331	1,59 %
70	6531	536	358	215	4,82 %	321	1,54 %
80	6410	507	508	198	4,44 %	309	1,48 %
85	6335	482	608	178	3,99 %	304	1,46 %
90	6170	458	797	170	3,81 %	288	1,38 %
95	5874	407	1144	142	3,18 %	265	1,27 %
99	5265	287	1873	73	1,64 %	214	1,03 %
99.9	4695	242	2488	38	0,85 %	204	0,98 %
99.99	3981	194	3250	12	0,27 %	182	0,87 %
99.9999	1470	85	5870	1	0,02 %	84	0,40 %
99.99999	1070	62	6293	1	0,02 %	61	0,29 %
99.999999	686	15	6724	1	0,02 %	14	0,07 %

Vedlegg C

Test av Attributter

- Signatur-id, gav oss som forventet et godt resultat ved testing av attributten. Dette henger sammen med at Snort-alarmer med enkelte signaturer sjelden opptrer som «Falsk Alarm», mens andre sjelden opptrer som «Reell Alarm».

- Signaturnavnet er det navn på hver signatur id. Det vil si disse attributtene logisk sett er like. Pga dette fører til sterk avhengighet mellom disse attributtene blir signatur navnet ikke tatt med. I seksjon 5.4 forklarer vi avhengighet mellom attributter nærmere.

- Sensor-id, i vårt tilfelle er Snort-alarmer hentet fra mange nett. Derfor er Snort-alarmene logget med en sensor id som sier noe om hvilke sensor disse stammer fra. Ved testing fikk vi forholdsvis dårlige resultater. Vi velger å ekskludere attributten da den bare sier noe om alarmshyppigheten på de ulike nett under perioden treningseksemplene dekker. Noe vi mener er en tilfeldig faktor.

- Signatur Prioritet, dette er en mulighet i snort til å gi signaturer ulik prioritet. I vår case var denne ikke i bruk. I tilfeller der denne er tilgjengelig vil vi anta dette kan være en god attributt til treningsettet, om den brukes fornuftig. En test på attributten slik vi har gjort for attributter, kan bekrefte eller avkrefte dette

- IP destination, gav bra resultater. Dette kan komme av at de ulike IP-ene er forskjellige maskiner som har forskjellig tjenester tilgjengelig. Noen tjenester kan være mer utsatt for angrep en andre, og dermed hjelpe klassifiseringen.

- IP time to live, gir oss gode resultater. Dette kan bl.a komme av hacker-verktøy som har faste verdier i dette header-feltet gir oss et mulig mønster å kjenne igjen.

- Signatur revisjon, attributten angir gjeldene versjonsnummer til signaturen. Eldre signaturer har gjerne vært gjennom flere endringer. Dette er ikke noe nyttig informasjon i vårt tilfelle, noe dårlige resultater ved testingen bekrefter. Attributten blir derfor fjernet fra treningssettet.
- IP header lengde, angir lengden på header feltet i IP-datagrammet. Gir forholdsvis dårlige resultater i test fordi lengden varierer sjelden. Vi velger likevel å benytte denne attributten da variasjoner fra standard-lengde på header kan indikere angrep.
- IP lengde. Gir ikke spesielt gode resultater. Likevel, attributten gir oss mulighet til å trene opp faste lengder av ip-datagrammer som kan bli generert av forskjellige verktøy. Dette kan i noen tilfeller hjelpe til å gjenkjenne en Snort-alarm som «Reell Alarm».
- IP flags. Mulige verdier her er svært få, så attributten gir sjelden avgjørende vekt under klassifisering. Likevel vil attributten ikke støye og blir derfor tatt med.
- IP type of service, våre tester ga svært dårlige resultater på denne attributten. Det kommer av at den er lik for nesten alle Snort-alarmene i vårt treningssettet. Vi valgte derfor å fjerne denne.
- IP protokoll, siden det er ip protokoll 4 som gjelder i dagens Internet var dette feltet helt uten variasjon, og har derfor ikke noen hensikt å inkludere i treningssettet.
- TCP destinasjon port, gir gode resultater ved testing. Dette skyldes at ekelte porter er mer utsatt for angrep en andre. Angrep fra hackerverktøy vil også lettere kjennes igjen ved at verktøyet benytter faste porter i angrepet.
- TCP window, gir oss nokså gode resultater ved teting, og blir benyttet.
- TCP reserved, er et reservert felt i TCP headeren. Dette ga svært dårlig resultater da det finnes lite variasjon her. Vi har derfor fjernet denne attributten fra treningssettet.
- TCP flagg, som med IP-flag er mulige verdier her få. Derfor gir denne attributten oss forholdsvis svake resultatet. Vi vil likevel benytte attributten da den øker muligheten til å fange opp på feil i TCP-segmentet.
- TCP options, benyttes for å avtale størrelsen på datamengden som skal kan sendes med i TCP datagrammene, som brukes blandt annet fordi ulike nett har ulik feilrate. Dette er informasjone som ikke hjelper i vår klassifiseringe, da vi fikk dårlige resultater når vi testet attributten. Denne blir derfor ekskludert fra treningsettet.

- TCP lenght, varierer lite, og kan derfor ikke gi spesielt gode resultater alene. Vil likevel kunne hjelpe, og vil ikke støye, så vi har valgt inkludere denne attributten.
- TCP urgent. Et ett-bits flag i tillegg til TCP-flag som sier noe om dette er ett urgent-segment eller ei. Da det som oftest har verdi lik null er det begrenset hvor stor nytteverdi attributten kan ha. Vi velger likevel å ta den med, da den ikke vil støye.
- UDP destionasjons-port. Som med TCP-destinasjonsport, gir dette oss en mulig til å gjenkjenne typiske porter benyttet i angrep, attributten er derfor inkludert i treningsettet.
- UDP lenght, taes med på samme grunnlag som TCP lenght.
- ICMP type, angir hvilken kontrollmelding dette er, noe som gir oss gode attributtverdier å trene på.
- ICMP code, ICMP-kode. Gir oss sammen med ICMP-type en fullstendig kontrollbeskjed, og er også en god attributt å trene på.

Vedlegg D

POPFile

Popfile er et verktøy som har stor suksess med å benytte maskinlærings prinsipper. Vi har studert POPFile for å få innspill og ideer til vår implementasjon, da særlig med fokus på lagring av læringsdata. Vi vil derfor kort forklare POPFile, som et eksempel på bruksområder av maskinlæring.

POPFile er et spamfilter som benytter en bayes basert metode for å klassifisere e-post. I sin enkleste form kjøres POPFile lokalt hos brukeren, og fungerer som et mellomledd (Proxy) mellom e-postklienten og e-postserveren. Programmet tar da imot forespørsler fra klienten, sender disse videre til tjeneren, tar i mot svaret og behandler det før det tilslutt sendes til klienten. Det finnes også mer eksperimentelle løsninger for å kjøre POPFile som SMTP-proxy, altså tilsvarende som i eksempelet over, men mellom til e-postservere. Programmet utvikles stadig og mulighet for mange atskilte brukere og bedre serverstøtte ligger i støpeskjeen.

D.1 Hvordan fungerer POPFile

POPFile benytter flere brukerdefinerte spann (Buckets) til å dele opp e-poster den har behandlet. Når POPFile først tas i bruk vet den ingenting om hva som skiller de ulike spannene fra hverandre og all e-post havner i et eget spann for uklassifisert post. Brukeren må så benytte et webgrensesnitt for å «lære» programmet hvor de ulike e-postene egentlig skulle havnet. E-postene deles opp i ord og ut i fra hvilke og hvor ofte et ord dukker opp i spannene kan POPFile bestemme sannsynligheten for at en ny e-post hører til i de ulike spannene. Skulle programmet gjøre en feil går brukeren inn og retter feilen, og over litt tid vil dermed POPFile læres opp til å kunne plassere stort sett all e-post i riktig spann.

D.2 Lagring av data

Erfaringene POPFile gjør (f.eks brukerens rettelser) lagres i en relativt enkel database. Fra versjon 0.21 benyttes enten SQLite [11] (standard innstillingen) eller MySQL, tidligere ble BerkleyDB benyttet med et litt annet oppsett enn det som er i dag. Kort fortalt lagres alle spann i en tabell, alle ord i en annen og i en tredje knyttes disse to tabellene sammen og et felt som teller antall tilfeller av ordet i spannet.